UNIVERSITY OF APPLIED SCIENCES



Diplomarbeit

Entwicklung eines Stereokamerasystems und eines Fernsteuersystems für den Sony Roboter AIBO ERS-7



Zur Erlangung des akademischen Grades

Diplom-Informatiker (FH)

Vorgelegt dem Fachbereich Mathematik, Naturwissenschaften und Informatik

der Fachhochschule Gießen-Friedberg

von

Michael Kreutzer

Januar 2005

Referent: Prof. Dr. -Ing. Axel Schumann-Luck
Koreferent: Prof. Dr. rer. nat. Klaus Wüst

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

Bereits in meiner Jugend, während der Ausbildung zum CNC-Programmierer, hatte ich ersten Kontakt mit computergesteuerten Werkzeugmaschinen und industriellen Robotern und war fasziniert von diesen Anlagen. Während des Studiums der Informatik, an der Fachhochschule Gießen-Friedberg, nutzte ich viele Gelegenheiten mich intensiver mit den Themen „Künstliche Intelligenz“ und „Autonome Roboter“ zu befassen. Ein anderes großes Interesse meinerseits ist die Grafische Datenverarbeitung. Als sich mir, auf der Suche nach einer passenden Aufgabe, an der Fachhochschule Gießen-Friedberg die Gelegenheit bot, ein Stereo-Bildverarbeitungssystem für einen Sony AIBO Roboter zu entwickeln, sah ich keinen Grund mehr zu zögern und so entstand diese Diplomarbeit.

Ich möchte an dieser Stelle allen meinen Dank aussprechen, die mich bei der Umsetzung dieser Arbeit unterstützten. Insbesondere Herrn Prof. Dr. -Ing. Axel Schumann-Luck, der mich als Diplomanden in seinem Team aufnahm, das mich hervorragend unterstützte. Natürlich möchte ich auch meinem Korreferenten, Herrn Prof. Dr. Klaus Wüst, danken. Meinen besonderen Dank möchte ich an Dipl. Phys. Martin Jung aussprechen, der mir stets mit Rat und Tat zur Seite stand und mir Zugang zu seiner Werkstatt gewährte. Nicht zuletzt möchte ich M. Eng., Dipl.-Ing. (FH) Jens Foerster für seine Hilfe und Unterstützung danken. Von ihm stammt zu einem wesentlichen Teil die Idee zum Thema dieser Arbeit.

Michael Kreutzer

Kapitel 1

Einleitung

Die moderne Technologie hat in nahezu allen Bereichen menschlichen Lebens unaufhaltsam Einzug gehalten. Eine Grenze der Miniaturisierung und der Leistungsfähigkeit von Computersystemen und anderen elektronischen Komponenten scheint in vielen Bereichen nicht absehbar. Das Gebiet der Robotik profitiert im großen Maße von dem technischen Fortschritt. So ist es heute möglich kleine und zugleich leistungsfähige Roboter zu konstruieren, die darüber hinaus mobil sind und sich in unbekannter Umgebung selbstständig zurechtfinden, also autonom sind.

Ein bekannter Stellvertreter der Gattung autonomer, mobiler Kleinroboter ist der AIBO ERS-7, ein von der Firma Sony entwickelter Roboterhund. Der Roboter zeigt autonomes Verhalten, künstliche Intelligenz und sogar künstliche Emotionen; er macht sein Verhalten also von inneren Stimmungen abhängig. Der ERS-7 wurde in erster Linie für den Massenmarkt produziert und dient dem Konsumenten als Unterhaltungsroboter. Aufgrund des hohen Entwicklungsstandes ist er aber ebenso für die Forschung als moderne und leistungsstarke Entwicklungsplattform einsetzbar.

1.1 Aufgabenstellung

Ein wesentliches Ziel dieser Arbeit ist es, die Sinnesorgane, genauer gesagt, den Sehsinn des AIBO ERS-7, durch die Entwicklung und Programmierung zusätzlicher Hardware, zu verbessern.

Dem Roboter fehlt die Fähigkeit des räumlichen Sehens. Das Tiefensehen ist eine sehr starke und wichtige Art der Wahrnehmung. Menschen und Tiere haben diese Fähigkeit bereits seit ihrer frühesten Entwicklung. Die griechische Mythologie deutet zwar auf die Existenz mystischer einäugiger Kreaturen, der Zyklopen, hin, tatsächlich sind der Wissenschaft aber nur sehr wenige Wesen bekannt, die über einen höherentwickelten Sehsinn, jedoch nicht über die Fähigkeit des stereoskopischen und somit räumlichen Sehens verfügen. Es liegt also auf der Hand, dass ein räumliches Sehvermögen für die Konstruktion eines hochentwickelten Roboters nicht nur eine hinreichende, sondern notwendige Bedingung ist.

Diese Diplomarbeit verfolgt neben der Umsetzung des Primärziels, der Entwicklung eines 3D Stereokamera-Systems für den AIBO ERS-7 noch ein sekundäres Ziel. Es soll einem Menschen ermöglicht werden in den kleinen AIBO Roboter hineinzuschlüpfen, seine Sinneswahrnehmungen zu erfahren und ihn durch möglichst natürliches menschliches Verhalten zu steuern.

Zur Umsetzung des Sekundärziels kommt moderne Virtual Reality Technik zum Einsatz, welche räumliches Hör- und Sehvermögen des kleinen Hundes für den Menschen erfahrbar macht. Zugleich werden dessen Bewegungen über eine Infrarotkamera erfasst um sie auf die Aktorik des Roboters zu übertragen.

1.2 Überblick über die Kapitel

Dem Leser soll in diesem Abschnitt ein Überblick über die einzelnen Kapitel vermittelt werden, um ihm das Suchen und Wiederfinden von Informationen zu erleichtern.

Kapitel 1, Einleitung: Beschreibt die Ziele und den Aufbau der Arbeit, sowie die Schriftkonventionen.

Kapitel 2, Grundlagen: Gibt eine Einführung in die Welt der autonomen Roboter und der Telerobotik, liefert eine Beschreibung des AIBO ERS-7 Roboters und vermittelt wichtige Grundlagen der Computerstereoskopie.

Kapitel 3, Stereokamera- und Fernsteuersystem: Beschreibt wichtige Konzepte, die zur technischen Realisierung der gestellten Aufgaben erarbeitet wurden.

Kapitel 4, Realisierung: Beschreibt zunächst die Einrichtung und Konfigurieren des Entwicklungsumfeldes. Im zweiten Teil des Kapitels wird die Programmierung eines Steuerungssystems für den AIBO ERS-7 Roboter dargelegt. Der dritte Teil des Kapitels behandelt die Realisierung von Programmmodulen zur Erfassung und Wiedergabe stereoskopischer Videodaten.

Kapitel 5, Schlussbetrachtungen: Eine kurze Zusammenfassung der Ergebnisse, Ausblicke auf weiterführende Anwendungen der entwickelten Systeme und neue Forschungsideen für zukünftige Projekte auf den beschriebenen Fachgebieten.

Zusätzlich wird die Arbeit durch eine **Begleit-CD** ergänzt. Sie enthält, neben diesem Dokument, den Quellcode des entstandenen Programms „Stereoview“, sowie das fertig compilierte Projekt als ausführbare Datei. Aufgrund des erheblichen Umfangs des Quellcodes, konnte dieser leider nicht als Anhang der Arbeit abgedruckt werden.

1.3 Schriftkonventionen

Zur Verbesserung der Strukturierung und Lesbarkeit des vorliegenden Dokumentes wird an dieser Stelle die Verwendung von Schrifthervorhebungen definiert.

Kursive Schrift wird verwendet für: Klassen-, Funktions- und Attributnamen.

Beispiele: *CDirect3D::Init(HWND hwnd), CDirect3D::ScrWidth = 800.*

Fett gedruckte Schrift wird verwendet für: Neu eingeführte Begriffe.

Beispiele: Wichtige Begriffe zum Verständnis des räumliche Sehen sind **Vergenz** und **Akkomodation**.

`Nichtproportionalschrift` wird verwendet für: Algorithmen bzw. Programmcode.

Beispiel:

```
for( int i=0; i<10; i++) {
  cout << „Hallo Welt!“;
}
```

<Einschluss> eines Begriffes in spitze Klammern wird verwendet zur: Bezeichnung von GUI[1]-Steuerelementen (Menüs, Buttons, usw).

Beispiele: <OK> Button, der Menüpunkt <Speichern>.

[1] GUI: Graphical User Interface.

Kapitel 2

Grundlagen

Im diesem Kapitel werden wichtige, zum Verständnis der Arbeit notwendige Grundlagen vermittelt.

Nach einem kurzen Ausflug in die Welt der autonomen Roboter und Teleroboter folgt eine detaillierte Beschreibung des Sony AIBO ERS-7. In der zweiten Hälfte des Kapitels werden die Grundlagen des räumlichen Sehens erläutert und gängige Geräte und Verfahren zur stereoskopischen Bildaufnahme und Darstellung beschrieben.

2.1 Autonome Roboter und Teleroboting

2.1.1 Autonome Roboter

Als **Autonomie** (griech. autonomia, „sich selbst Gesetze gebend“, „“selbständig“) bezeichnet man „Selbständigkeit“, „Unabhängigkeit“, „Selbstverwaltung“ oder „Entscheidungsfreiheit“. [1]

Demnach sind **autonome Roboter** also Maschinen, die selbständig handeln können und dabei inneren Gesetzen gehorchen. Eine solche Maschine, einmal aktiviert, handelt nach zuvor eingegebenen Verhaltensvorschriften und bedarf während des Betriebes keiner eingreifenden Steuerung durch einen Menschen.

Mobile autonome Roboter sind eine besondere Untergruppe der autonomen Roboter. Ein Roboter wird als **mobil** bezeichnet, wenn er Aktoren zur freien

Veränderung seiner Position in der Umwelt besitzt. Der Aktionsraum eines solchen Roboters wird in der Regel nur durch Umgebung (Wände) und begrenzte Energieressourcen beschränkt. Die Orientierung in der Umwelt muss daher durch externe Sensoren erfolgen (vergleiche [3]).

Um frei in einer möglicherweise unbekannten Umgebung navigieren und agieren zu können, müssen Informationen, die über die externen Sensoren aufgenommen wurden, im Roboter analysiert und verarbeitet werden. Durch entsprechende Auslegung eines Roboter-Steuerprogramms, kann ein mobiler autonomer Roboter selbständig auf die Gegebenheiten seiner Umwelt reagieren, in ihr navigieren und einfache Aufgaben erfüllen.

Die Anwendungsgebiete mobiler autonomer Roboter sind vielfältig. Interessante Einsatzgebiete sind die automatische Überwachung von Gebäuden und die automatisierte Inspektion. Mobile Roboter können selbständig Gänge und Räumlichkeiten befahren und auf ungewöhnliche Begebenheiten oder Veränderung ihrer Umgebung reagieren, beispielsweise durch das Auslösen eines Alarms, wenn sich eine erfasste Person nicht authentifizieren kann. Es bietet sich an, routinemäßige Inspektionsarbeiten, beispielsweise in Maschinenparks, durch mobile autonome Roboter ausführen zu lassen. Im industriellen Bereich werden mobile Roboter außerdem immer häufiger als intelligente Transportsysteme eingesetzt. Eine Produktionsmaschine, die nicht mehr über ausreichend Rohmaterial verfügt, kann von einem autonomen mobilen Roboter bei Bedarf „nachgeladen“ werden. Insbesondere für Arbeiten in schwer zugänglichen Bereichen, wie dem Weltraum, in denen nur bedingt eine Kommunikation zu einem elektronischen System aufgebaut werden kann (Funkschatten, Signalverzögerung durch große Entfernungen im Weltall), finden mobile autonome Roboter ihre besondere Verwendung.

Für alle anderen Anwendungsgebiete muss fallspezifisch entschieden werden, ob ein autonomer Roboter die Aufgaben zufriedenstellend erledigen kann oder, ob in gewissen Situationen ein „gesunder Menschenverstand“ eingreifen muss, um das Verhalten des Roboters zu steuern oder zumindest zu kontrollieren. In diesen Fällen empfiehlt sich der Einsatz eines sogenannten **Teleroboters**, der nur in bestimmten Bereichen autonom agiert.

2.1.2 Teleroboting

Teleroboter sind Roboter, die eine Kommunikationsverbindung, beispielsweise eine Funkverbindung, zu einem menschlichen Operator besitzen. Dieser kann den Roboter steuern oder kontrollieren und bei Bedarf in den autonomen Ablauf eingreifen. Für alle Abläufe, während denen das Eingreifen eines Menschen sinnvoll ist und eine Kommunikationsverbindung aufgebaut werden kann, besitzen Teleroboter Vorteile gegenüber rein autonomen Robotern. Ein interessantes Anwendungsgebiet für Teleroboter ist unter anderem die Automatisierungstechnik. Häufig werden hier große Produktionslinien, bestehend aus einer Vielzahl hintereinandergereihter Robotersysteme, aufgebaut. Ein menschlicher Operator kann, von einem festen Arbeitsplatz, steuernd in die Abläufe mehrerer teilautonomer Roboter eingreifen und diese überwachen.

In Anbetracht dessen, das der Mensch nicht vor Ort sein muss, um einen Teleroboter zu steuern, eignen sich selbige gut für den Einsatz in lebensgefährdenden oder unzugänglichen Gebieten (Mienenräumung, Aufräumarbeiten nach atomaren oder chemischen Unfällen). Ein sehr bekanntes Beispiel für ein teilautonomes Teleroboter-System ist die Marssonde „Pathfinder“. In der im April 1996 gestarteten Mission gelang es der NASA, eine Sonde, bestehend aus einer Landestation und dem etwa 16 kg schweren Rover „Sojourner“ (Abbildung 2.1), auf dem Mars abzusetzen. Der mobile Rover diente im Wesentlichen als Trägerplattform für ein „Alpha-Proton - Röntgenstrahlenspektrometer“, einem Messinstrument zur elementaren Untersuchung von Oberflächenmaterialien. Darüber hinaus verfügte der Rover über zwei Kameras (Front- und Heckkamera). Gesteuert wurde der Marsrover über Funkverbindung zur Landestation, welche ihrerseits mit der Bodenstation auf der Erde kommunizierte. [4]

Die große Entfernung zwischen Mars und Erde führt zu einer erheblichen Verzögerung der Funksignale zwischen Bodenstation und Mars-Sonde. Im Idealfall würden deshalb nur Steuerbefehle höherer Ordnung an ein dort operierendes Vehikel übertragen. Die Zeit zwischen den eintreffenden Funksignalen könnte der Roboter nutzen, um die übergeordneten Befehle intelligent und selbständig auszuführen. Ein solcher Befehl könnte beispielsweise die Anordnung eines Positionswechsels eines Mars-Rovers sein. Wie der Rover seine neue Position erreicht oder

auftretende Probleme, wie das Umfahren eines Felsens, löst, bliebe ihm selbst überlassen (Teilautonomität).

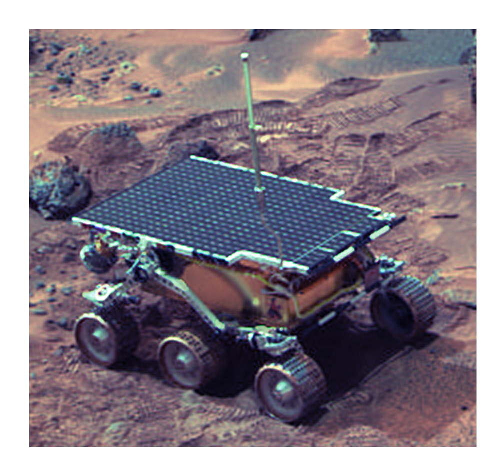


Abbildung 2.1: Marsrover „Sojourner“ [4]

2.2 AIBO ERS-7

Der Name „**AIBO**“ ist die Produktbezeichnung eines, von der Firma Sony entwickelten, Roboterhundes. Der **AIBO ERS-7** vertritt bereits die dritte Generation dieser Serie. Die Bezeichnung „AIBO“ steht für „Artifical Intelligence Robot“ (dt. Roboter mit künstlicher Intelligenz). Im englischsprachigen Kontext ist „Artifical Intelligence“ auch als eine Anspielung auf einen „Roboter mit Augen“ zu verstehen. In Japan, der Geburtsstätte des kleinen Roboterhundes, bedeutet AIBO soviel wie „Kumpel“ oder „Partner“. (Vergleiche [5, 6])

2.2.1 Idee

Nach eigenen Angaben hat Sony mit dem ERS-7 ein künstliches Wesen erschaffen, das sich nicht nur frei bewegen, sondern auch selbständig denken kann. Der ERS-7 kann Gefühle ausdrücken und instinktives Verhalten zeigen. Und er kann seine Fähigkeiten und Persönlichkeit durch Lernen weiterentwickeln. Unter **„Lernen“** ist in diesem Zusammenhang die Ausbildung und Änderung von Verhalten durch äußere Einflüsse zu verstehen. Nach obiger Definition (Abschnitt 2.1.1) ist

der AIBO ERS-7, unter Verwendung der von Sony mitgelieferten **„AIBO Mind“ Software**, ein mobiler autonomer Roboter. Die Software setzt sich aus drei, miteinander kombinierten, Modulen zusammen: **AIBO Recognition, AIBO Life 2 und AIBO Explorer**. Mittels dieser Programmmodule wird der Roboter in die Lage versetzt, Gegenstände und Töne wiederzuerkennen, Instinkte und Gefühle eines lebendigen Lebewesens zu simulieren und selbständig seine Umgebung zu erkunden.

AIBO kommuniziert mit dem Menschen, indem er Gefühle zeigt und Verhaltensweisen zur Schau stellt, welche auf Informationen basieren, die er in seiner Umgebung gesammelt hat oder die ihm durch seinen menschlichen Partner vermittelt wurden. Auch wird sein Verhalten, ähnlich wie bei reellen Lebewesen, durch eine Reihe von grundlegenden Instinkten beeinflusst.

Diese Instinkte ordnen sich in:

Zuneigungsinstinkt: Das Bestreben mit Personen Kontakt aufzunehmen.

Suchinstinkt: Das Bestreben Neugier zu befriedigen.

Bewegungsinstinkt: Das Bestreben sich in seiner Umgebung zu bewegen.

Ladeinstinkt: Das Bestreben eine Stromquelle zum Laden des Akkus zu finden, entsprechend dem menschlichen Hungergefühl.

Schlafinstinkt: Das Bestreben nach anstrengenden Aktivitäten oder langer Betriebszeit auszuruhen.

Neben den Instinkten wird das Verhalten des AIBO durch Gefühle und Stimmungen beeinflusst, welche er im Zusammensein mit Menschen zur Schau stellt. Basierend auf vorhergehenden, äußeren Ereignissen oder den, oben erläuterten, Instinkten bringt AIBO Stimmungen wie Fröhlichkeit, Ärger oder Trauer zum Ausdruck. Er vermittelt diese Stimmungen über seine Körpersprache, über Töne und über die Form und Farbe seiner Augen, welche durch eine Reihe mehrfarbiger LEDs eindrucksvoll dargestellt werden. Das Zusammenspiel von Instinkten und die daraus resultierenden Stimmungen, bilden die Persönlichkeit des kleinen Hausroboters.

AIBO ist in der Lage sich im Verlaufe seines Roboterlebens weiterzuentwickeln. Dazu passt er seine Verhaltensweisen den Begebenheiten seiner Umgebung an. Er simuliert den psychischen Prozess der **Konditionierung**, den wir von realen Lebewesen kennen. Das heißt, er lernt durch Erfolg und Misserfolg. Führt ein bestimmtes Verhalten zu einem Ziel oder einer Belohnung - er findet seine Ladestation um seinen „Hunger" zu stillen oder sein Besitzer lobt ihn - so führt dies zu einer positiven Konditionierung. Er wird dieses Verhalten in Zukunft öfter zeigen. Im Gegensatz dazu wird er ein Verhalten, dass ihn nicht zum Ziel führt oder eine Bestrafung zur Folge hat, in Zukunft weniger häufig zeigen. Er „verlernt" dieses Verhalten.

Um sich in seiner Umgebung zurechtzufinden und zu kommunizieren, verfügt der ERS-7 über zahlreiche Sensoren und Aktoren. Er kann, durch eine im Kopf eingebaute Kamera, „sehen" und Objekte, sogar Gesichter von Menschen wiedererkennen und verfolgen. Auch ist er in der Lage, über eingebaute Mikrofone, Audioinformationen aufzunehmen und Geräusche, sowie Sprache zu erkennen und eine Vielzahl von Wortbefehlen zu identifizieren. Auch verfügt er über einen Beschleunigungssensor, der ihm ermöglicht auf vier Beinen sicher zu laufen. Zur Erkennung von Hindernissen oder Abgründen stehen zwei Infrarot-Entfernungsmesssensoren bereit. Einer davon sitzt im Kopf des Roboterhundes und misst die Entfernung zu dem Objekt, das sich im Fokus der Kamera befindet. Der zweite Sensor, in der Brust, ist im 60°-Winkel nach unten gerichtet und dient in erster Linie dazu, AIBO vor nahenden Kanten oder Treppenabgängen zu warnen. Um Berührungen seines Besitzers wahrnehmen zu können, verfügt AIBO über drei Berührungssensoren am Rücken und einen an seinem Hinterkopf.

Verschiedene Aktoren ermöglichen dem Roboter sich zu bewegen und seine Körpersprache auszudrücken. Vier Beine mit beweglichen Schulter/Hüft- und Ellenbogen/Kniegelenken ermöglichen ihm das Laufen. Drei Gelenke am Kopf ermöglichen das Drehen und Neigen desselben. Ein wichtiges Element zum Ausdruck von Gefühlen und Stimmungen eines Hundes ist der Schwanz. Ihm wurden beim AIBO ERS-7 zwei Gelenke spendiert.

Sony stellt für den AIBO ERS-7 kostenlos ein umfassendes Entwicklungskit, das „**AIBO SDK**"[1], bereit. Es besteht aus den Komponenten:

[1] Eine nähere Beschreibung der SDK-Komponenten findet sich auf den Sony AIBO Entwickler Webseiten [6].

Open-R SDK: Eine umfassende Entwicklungsumgebung für das Cross-Platform-Development[2].

R-Code SDK: Eine, vor allem für Nichtspezialisten geeignete, einfache Skriptsprache.

AIBO Remote Framework SDK: Eine auf der Programmiersprache „C++“ basierende Entwicklungsumgebung zum Entwurf von PC-Anwendungen zur Steuerung des AIBO via WLAN über einen PC.

AIBO Motion Editor: Ein Programm zur Editierung von Bewegungsabläufen, welche auf dem AIBO wiedergegeben werden können.

Aufgrund dieser umfassenden Hardwareausstattung, sowie der Vielzahl inkludierter Sensoren und der vollständigen, von Sony freigegebenen Entwicklungsumgebung „AIBO SDK“ stößt das Produkt AIBO auf großes Interesse in der Robotikforschung und vor allem - durch die relativ geringe Anschaffungskosten - im universitären Umfeld.

2.2.2 Technische Spezifikation

Das Herz des AIBO ERS-7 bildet ein 64 Bit Risc-Prozessor-System mit 64 Megabytes internem Speicher. Als Programm-Medium werden speziell auf den AIBO abgestimmte Sony Memorysticks verwendet. Die beigelegte AIBO-Mind Software wird auf einem 32 MB Stick geliefert, während der separat erhältliche Programmierstick lediglich über 16 MB Speicherkapazität verfügt.

Der AIBO ERS-7 verfügt über eine Reihe beweglicher Extremitäten und Körperteile: Kopf, Maul, Beine, Ohren und Schwanz. Insgesamt bilden die beweglichen Teile 20 Freiheitsgrade (Tab. 2.1).

An der Unterseite des ERS-7 befinden sich die Ladekontakte, sowie jeweils ein Schalter zum Einstellen der Lautstärke und zur Aktivierung/Deaktivierung des WLAN-Moduls. Die Bildaufnahme erfolgt über einen in der Nase eingebauten CMOS-Bildsensor mit einer Auflösung von 350.000 Bildpunkten. Zur Aufnahme

[2]Entwicklung von Software für eine, von der Erstellungsplattform, abweichende Zielplattform. Hier: PC als Erstellungsplattform, AIBO als Zielplattform.

Tabelle 2.1: Freiheitsgrade AIBO ERS-7 [5]

Körperteil	Freiheitsgrade
Kopf	3
Maul	1
Beine	3 x 4
Ohren	1 x 2
Schwanz	2

von Tönen besitzt der ERS-7 zwei Mikrofone, eines auf jeder Seite des Kopfes. Die Wiedergabe von Tönen erfolgt über einen in der Brust eingebauten Lautsprecher.

Der Roboter verfügt über diverse interne Sensoren: Zwei Infrarot-Abstandssensoren, einen Beschleunigungssensor und einen Schwingungssensor. Extern verfügt er über: Einen Kopfsensor, drei Rückensensoren, einen Kinnsensor und Tastsensoren an allen vier Pfoten.

Die Leistungsaufnahme des ERS-7 liegt bei durchschnittlich 7 Watt, womit bei voll geladenem Akku eine Betriebsdauer von ca. 1,5 Stunden einhergeht.

Zur Kommunikation mit der Außenwelt besitzt der AIBO ERS-7 ein eingebautes WLAN-Modul. Dieses ist Wi-Fi-zertifiziert und kompatibel mit den IEEE 802.11/IEEE 802.11b Standards. Durch entsprechende Konfiguration können die Wireless-Kanäle 1-11 auf dem 2,4 GHz Frequenzband verwendet werden. Auf Wunsch kann eine Verschlüsselung nach dem WEP 64 oder WEP 128 Standard eingesetzt werden. [6, 5]

2.3 3D Stereoskopie

Die folgenden Abschnitte sollen einen Überblick über das Thema „3D Stereoskopie“ vermitteln. Dazu wird der Vorgang des räumlichen Sehens am Beispiel des Menschen erläutert und einige Geräte zur Aufnahme und Wiedergabe stereoskopischer Bilddaten beschrieben. Zuletzt werden die beiden wichtigsten Verfahren zur Aufbereitung eines Stereo-Videosignals erklärt.

2.3.1 Visuelle Wahrnehmung des Menschen

Die Augen gehören zu den wichtigsten Sinnesorganen des Menschen. Personen, die unter dem Verlust des Augenlichts leiden, haben in der Regel Schwierigkeiten ihre Umwelt vollständig wahrzunehmen und sich in ihr zu orientieren.

Die Fähigkeit des Menschen seine Umwelt räumlich wahrzunehmen ermöglicht ihm die präzise und schnelle Abschätzung von Entfernungen zu Objekten in seiner Umwelt. Der urzeitliche Menschen wurde so zum erfolgreichen Jäger. Der moderne Mensch kann durch den Einsatz seiner Tiefensehkraft filigranste Maschinen bauen oder Statuen aus einem Stein schlagen ohne sich mit dem Hammer seine Hand zu verletzen.

Das räumliche Sehen ist ein wesentlich stärkerer Sinneseindruck, als das zweidimensionale Sehen, welches wir vom Betrachten von Abbildungen auf Papier oder dem Computerbildschirm kennen. Daher ist die Darstellung möglichst echter Dreidimensionalität in allen grafischen Anwendungen, in denen die reale oder auch eine fantastische Welt abgebildet werden soll, erstrebenswert.

Die Grundlage für die Entwicklung stereoskopischer Aufnahme- und Darstellungsverfahren ist das Verständnis der Erfassung dreidimensionaler Bilder durch die visuellen Sinnesorganen des Menschen und der Wahrnehmung durch das menschliche Gehirn.

Das Auge gleicht einer Fernsehkamera, mit dem Unterschied, dass die lichtempfindlichen Sehsinneszellen (Zäpfchen und Stäbchen) nicht gleichmäßig über die gesamte Netzhaut verteilt sind. An der Stelle des schärfsten Sehens, der **Fovea**, beträgt die Auflösung der Sehsinneszellen - unter Einbezug der Augenoptik - etwa 1/60°. Nach außen hin fällt die Sehschärfe auf 1/10 dieses Wertes ab. In der Netzhaut finden sich zwei Kategorien von Sehsinneszellen: **Stäbchen und Zäpfchen**. Die Stäbchen sind in relativ geringer Dichte eher im äußeren Bereich der Netzhaut angeordnet. Sie haben im Vergleich mit den im Zentrum angeordneten Zäpfchen eine höhere Lichtempfindlichkeit und reagieren schnell auf Helligkeitsänderungen; erfassen also schnelle Bewegungen oder Flimmern. Die Zäpfchen sind in großer Dichte im inneren Bereich der Netzhaut angeordnet und bilden dort den Bereich des schärfsten Sehens. Es existieren drei unterschiedliche Arten dieser Sehsinneszellen, die auf unterschiedliche Wellenlängen des Lichts ansprechen

(Farbsehen). Sie haben eine geringere Lichtempfindlichkeit als die Stäbchen und reagieren weniger schnell auf Helligkeitsänderungen.

Die von den Sehsinneszellen aufgenommenen Informationen werden über Nervenzellen zu den **Ganglienzellen** weitergeleitet. Deren ca. 1 μm dicken Fortsätze bündeln sich zum Sehnerv und verlassen das Auge im „blinden Fleck“, ca. 15° neben der Stelle des schärfsten Sehens. Beide Sehnerven laufen dann zur partiellen Sehbahnkreuzung, dem **Chiasma**.

Das Chiasma spielt eine wesentliche Rolle für das dreidimensionale Sehen. Dort werden die rund 1 Million Nervenfasern sortiert. Fasern von Sehzellen, die Bildinformationen links der **Fixierlinie,** diese beschreibt die Mitte des Blickfeldes, liefern, werden in die rechte Gehirnhälfte weitergeleitet. Fasern mit Informationen rechts der Fixierlinie in die linke Gehirnhälfte. Bei Ankunft der Fasern im **primären visuellen Cortex** (Sehzentrum) müssen diese so sortiert sein, das die Fasern korrespondierender Netzhautzellen an der gleichen Stelle im Sehzentrum ankommen. **Korrespondierende Netzhautzellen** bedeutet, genau die Paare von Netzhautzellen des linken und rechten Auges, auf denen gleiche Punkte eines Motivs in der **Fixationsentfernung** (Entfernung eines anvisierten Punktes) beider Augen abgebildet werden.

Nimmt man an, die Augen einer Testperson fixierten die dünne Spitze eines Bleistiftes, so korrespondieren genau jene beiden Netzhautzellen, die auf der Netzhaut des linken bzw. rechten Auges die Spitze des Bleistiftes abbilden.

Um ein Motiv scharf abzubilden wirken zwei Regelmechanismen: Die **Vergenz** und die **Akkomodation**. Die Vergenz steuert über Muskeln die Sichtachsen beider Augen so, dass ein in der Fixationsentfernung befindliches Motiv bei beiden Augen auf den korrespondierenden Netzhautzellen abgebildet wird (Abbildung 2.2). Akkommodation bezeichnet den Vorgang des Einstellens der Augenlinse, so, dass ein Motiv bei wechselnder Entfernung scharf auf der Netzhaut abgebildet wird. Vergenz und Akkommodation wirken bei normaler Entwicklung des Sehens immer zusammen.

Wir nehmen an, ein Motiv wurde durch die Augen - mittels Vergenz und Akkommodation - korrekt fixiert. (Abbildung 2.3 zeigt schematisch eine entsprechende Versuchsanordnung; Abbildung 2.4 die resultierenden, durch die Augen aufgenommenen Abbilder der Szene.)

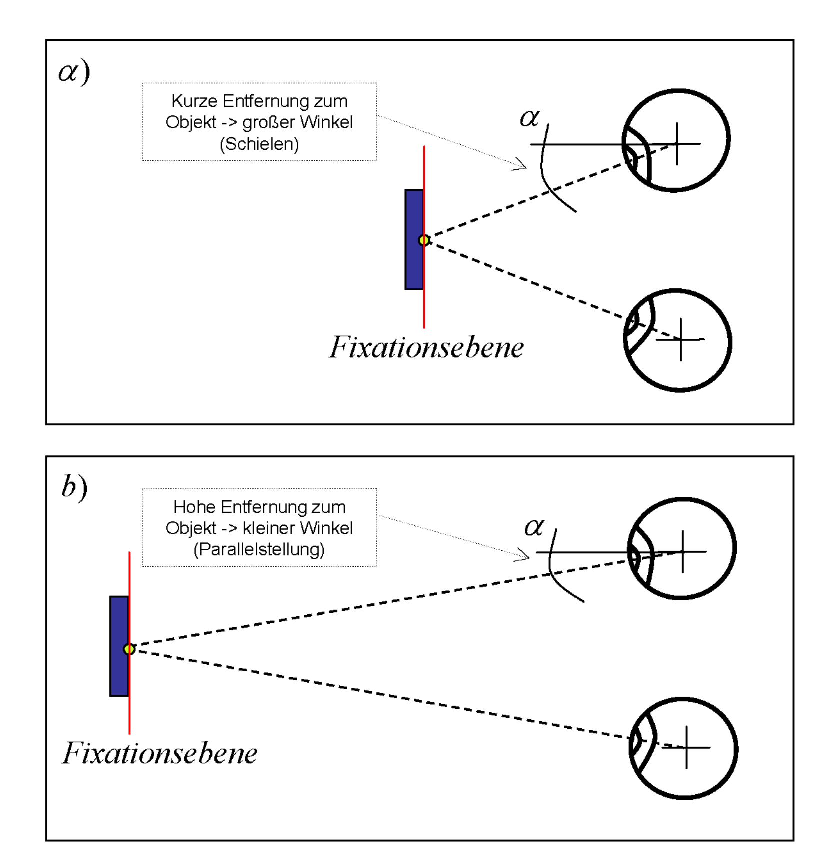


Abbildung 2.2: Einstellung der Vergenz

Bilder von Motiven, die näher als die Fixationsentfernung liegen sind gegenüber den korrespondierenden Netzhautzellen nach außen versetzt. Für Bilder, die hinter der Fixationsentfernung liegen, gilt das Umgekehrte: Sie sind nach innen versetzt. Den seitlichen Versatz nennt man **Disparität**.

An dieser Stelle ist es den Neuronalen Netzen des Sehzentrums möglich, die Signale der beiden Augen zu „verrechnen“. Signale von korrespondierenden Netzhautzellen aktivieren „Fixationsneurone“. Dahingegen regen Signale, die von leicht verschobenen Netzhautzellen kommen „Nah-“ oder „Fernneurone“ an, je nachdem, in welcher Richtung die Bilder auf der Netzhaut verschoben sind.

In Computeranwendungen die stereoskopische 3D-Grafik darstellen wird in der Regel nur die Vergenz, also die relative Verschiebung eines Motivs zwischen den

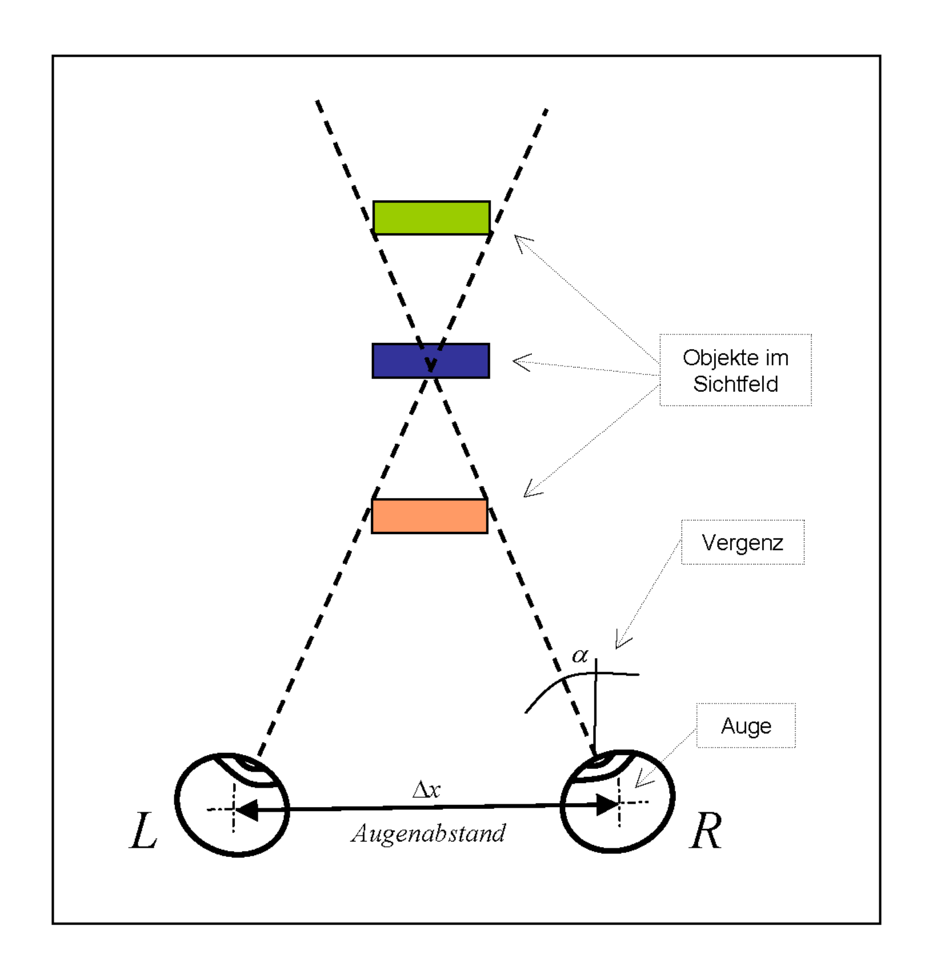


Abbildung 2.3: Fixierung eines Motivs durch ein Augenpaar

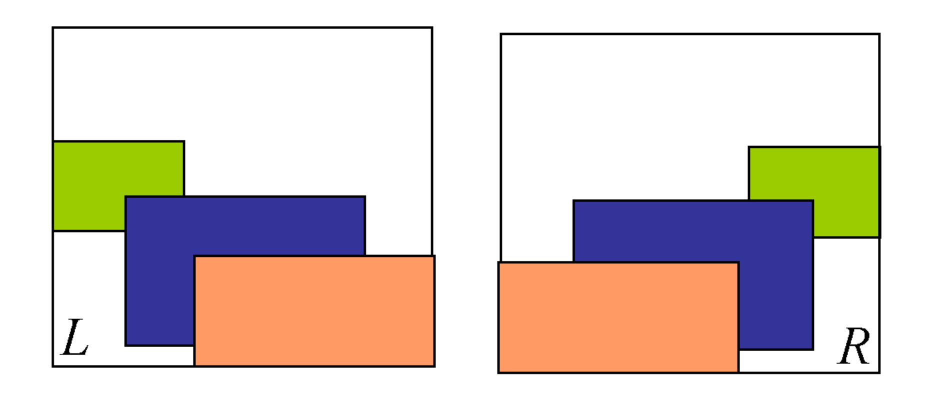


Abbildung 2.4: Disparität

beiden Augen, simuliert. Dazu wird ein 3D-Modell zweimal gerendert[3] und zwar aus der Perspektive jedes der beiden Augen. Die Simulation der Akkummodation, des „Scharfstellens“ der Augenlinsen, erfordert einen sehr hohen Hardwareaufwand und führt zu keiner wesentlichen Verbesserung des Darstellungsergebnisses. Auf die Simulation der Akkommodation wird daher in der Regel verzichtet. [7]

2.3.2 Stereoskopische Aufnahmesysteme

Zur Erfassung von 3D-Stereo-Bildern, die nicht aus künstlich erzeugten Vektordaten reproduziert werden, sondern der realen Welt entspringen, werden stereoskopische Bildaufnahmeverfahren eingesetzt. Auf dem Markt existiert eine Vielzahl von Geräten, die dem Anwender die Aufnahme von Stereobildern oder Videos ermöglichen sollen.

Eine naheliegende Lösung ist der Einsatz einer **Doppelkamera**. Ein solches Gerät wird nicht für den Massenmarkt angefertigt und ist daher sehr teuer. Aus diesem Grund wird die Doppelkamera häufig aus zwei handelsüblichen, im Augenabstand von etwa 6 cm angeordneten, Foto- oder Videokameras konstruiert. Abbildung 2.5 zeigt eine aus zwei Canon Elura DV-Camcordern zusammengesetzte Stereokamera. Die Auslöser der Kameras müssen über die Eingänge für die Fernauslösung synchronisiert werden. Die, auf den beiden Videobändern aufgezeichneten, Bilddaten werden auf einem Computer nachbearbeitet und zu einem stereoskopischen Video zusammengefügt.



Abbildung 2.5: Doppelkamera, 2 x Canon Elura mini-DV Camcorder [8]

[3]Rendern im Sinne der 3D-Grafik: Zeichnen und Texturieren einer dreidimensionalen Vektorgrafik zur Darstellung auf einem Bildschirm.

Es ist auch möglich eine Fotokamera, horizontal verschiebbar, auf einer geraden Metallschiene zu montieren. Die Bilder des linken und rechten Auges werden nacheinander manuell aufgenommen. Diese Methode macht es jedoch unmöglich, sich bewegende Motive zu erfassen und ist daher nur für Landschaftsaufnahmen oder „erste Versuche“ geeignet.

Eine weitere, auch für bewegte Bilder geeignete, Variante ist der Einsatz eines **Prismenvorsatzes**. Die Speziallinse wird vor das Objektiv einer Kamera platziert und teilt das aufgenommene Bild an der vertikalen Mittellinie in zwei Halbbilder, von denen jedes aus einer etwas anderen Perspektive aufgenommen wurde. Nachträglich wird das in horizontaler Richtung gestauchte „Stereobild“ entzerrt (Abbildung 2.6).

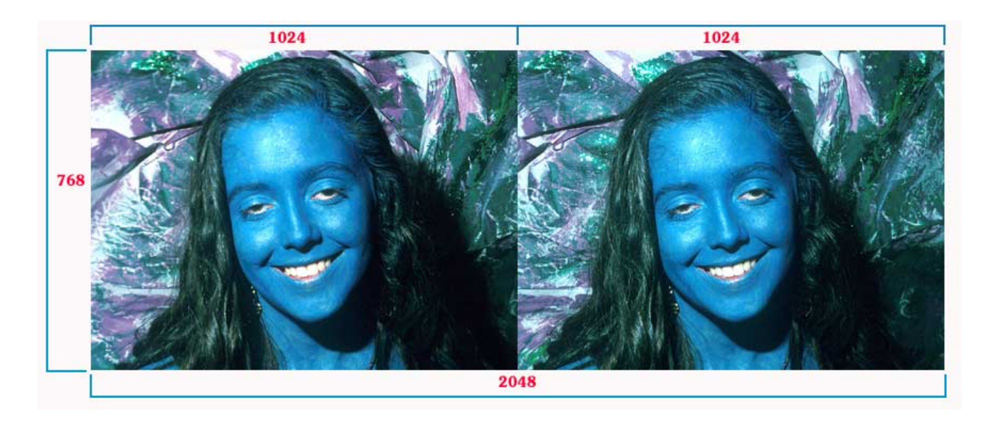


Abbildung 2.6: Entzerrte Aufnahme mit Prismenvorsatz [8]

In der linken und rechten Bildhälfte werden zwei, aus leicht unterschiedlichen Perspektiven aufgenommene, Aufnahmen des Motivs abgebildet.

Eine letzte, recht interessante Methode zur Aufnahme von stereoskopischen Videobildern ist der Einsatz eines **LC-Shuttervorsatzes** (Abbildung 2.7).

Der Adapter enthält eine Spiegelkonstruktion, die es ermöglicht ein Motiv aus zwei leicht unterschiedlichen Perspektiven aufzunehmen [9]. Beide Strahlengänge werden mittels Prisma oder halbdurchlässigem Spiegel überlagert und dem Kameraobjektiv zugeführt. Damit sich die Bilder der beiden Perspektiven nicht mischen, wird in jeden der beiden Strahlengänge ein sogenannter **LC-Shutter** eingesetzt. Der aus Flüssigkristallen (engl. **L**iquid **C**rystal) aufgebaute Verschluss (engl. Shutter) ist im Normalzustand durchsichtig. Sobald eine Spannung an den



Abbildung 2.7: LC-Shutter-Vorsatz für Videokameras [9]

Kristallen angelegt wird, verdunkelt sich der Shutter und blockiert den Strahlengang. Das Bild einer Fernsehkamera (nach PAL-Norm) wird aus zwei, zeitlich versetzt aufgenommenen Halbbildern aufgebaut. Zuerst werden alle ungeraden Bildzeilen aufgenommen, 1/50 Sekunde später alle geraden Zeilen. Synchronisiert man nun die beiden LC-Shutter über den Videoausgang der Kamera mit dem PAL-Signal, so wird es möglich, jeweils eine Perspektive in einem Halbbild des Videosignals abzulegen. Statt eines normalen Videosignals mit 25 Vollbildern pro Sekunde in PAL-Auflösung erhält man auf diese Weise ein Signal, welches 25 Stereobilder pro Sekunde mit halbierter vertikaler Auflösung überträgt. Dieses Signal lässt sich, bei der Darstellung auf einem Fernsehgerät, mittels einer LC-Shutterbrille, mit geringem technischen Aufwand in die Abbildungen der zwei ursprünglich aufgenommenen Perspektiven splitten. Das linke Auge des Betrachters erhält die Informationen des ersten Halbbildes, das rechte die des zweiten. Die oben beschriebene Methode ist besonders interessant, da sie sich mit relativ geringem Aufwand, durch Erweiterung herkömmlicher Videohardware, kostengünstig umsetzen lässt. Sie hat allerdings den Nachteil, dass sich, unter Einsatz einer LC-Shutterbrille, Bildhelligkeit und Bildwiederholrate halbieren.

2.3.3 Stereoskopische Darstellungssysteme

Das zentrale Problem bei der Darstellung stereoskopischer Bilddaten liegt in der Aufteilung der Teilbilder auf das linke und rechte Auge des Betrachters. Herkömmliche Bildschirme und Projektoren können ohne den Einsatz spezieller Techniken und Methoden lediglich „zweidimensionale“ Bilder wiedergeben.

Eine alte, einfache Technik zur Darstellung stereoskopischer Bilder ist der **Schaukasten**. In einem kleinen Kasten, der an seiner Vorderseite mit zwei Löchern versehen ist, werden im Augenabstand zwei Dia-Positive mit den Teilbildern der Stereoaufnahme platziert. Hinter den Bildern befindet sich eine Lichtquelle zur Durchleuchtung. Der Schaukasten ist einfach und effektiv, da sich mit ihm sogar Farbaufnahmen problemlos darstellen lassen. Er entspricht jedoch nicht mehr den Anforderungen der modernen Video- und Computertechnik.

Eine, auch heute noch, weit verbreitete Methode zur Trennung der Stereobildkanäle ist die **Anaglyphen-Darstellung**. Die beiden Einzelbilder der Stereoaufnahme werden durch die Verwendung unterschiedlicher Farbkanäle kodiert. Beispielsweise Rot als Farbkanal für das linke Auge und Cyan als Farbkanal für das rechte Auge. Abbildung 2.8 zeigt ein mit dieser Technik dargestelltes Stereobild. Eine Brille mit Farbfilterfolien (Abbildung 2.9) trennt die beiden Stereokanäle wieder auf. So werden Bildinformationen des linken und rechten Teilbildes korrekt an die beiden Augen des Beobachters weitergeleitet. Die Farbe Cyan könnte man, etwas anders ausgedrückt, als Grün-Blau bezeichnen. Beide Filter einer Rot/Cyan-Filterbrille, zusammengenommen, können also von rotem, grünen und blauem Licht durchdrungen werden. Durch richtiges Mischen dieser drei Farben lassen sich alle, vom menschlichen Auge wahrzunehmenden Farbtöne, kreieren. Man muss dazu allerdings voraussetzen, dass das menschliche Gehirn die rot eingefärbten Bildinformationen auf dem linken Auge mit den Grün-Blau gefilterten Bildinformationen des rechten Auges wieder zu einem Farbbild zusammensetzt. In der Praxis führt diese Technik zu eher unbefriedigenden Ergebnissen. Allerdings können alle „farbigen“ Wiedergabemedien, ohne technische Änderungen, verwendet werden. Auf diese Weise können Bildbände oder 3D-Fernsehprogramme für den Massenmarkt erstellt werden.

Eine, gegenüber der Anaglyphendarstellung, sehr viel bessere Technik ist die Stereokanalkodierung mittels **linearer Polarisationsfilter**. Für das bessere Verständnis dieser Technik wollen wir das Licht als elektromagnetische Wellen interpretieren. Elektromagnetische Wellen sind Transversalwellen. Das heißt, sie besitzen einen Schwingungsvektor, der orthogonal zu ihrer Ausbreitungsrichtung ausgerichtet ist. Man kann sie mit Wellen auf einer Wasseroberfläche vergleichen. Während sich die Wellen in einer bestimmten Richtung, entlang der Oberfläche, ausbreiten, schwingt diese nach oben und unten. Der Schwingungsvektor steht



Abbildung 2.8: Rot/Cyan Anaglyphen-Darstellung [8]



Abbildung 2.9: Rot/Cyan Farbfilterbrille [8]

hier genau orthogonal, sowohl zur Ausbreitungsrichtung, als auch zur Richtung der Schwerkraft. Da die Schwerkraft auf Lichtquanten nur eine geringe Wirkung hat, steht der Schwingungsvektor der Lichtquelle zwar orthogonal zur deren Ausbreitungsvektor, ist jedoch um diesen frei drehbar. Den Drehwinkel des Schwingungsvektors, nennt man Polarisation des Lichts. Herkömmliche Lichtquellen, liefern Lichtwellen mit einer Mischpolarisation. Durch den Einsatz von sogenannten linearen Polarisationsfiltern ist es möglich Wellen mit einer bestimmten Polarisation herauszufiltern (Abbildung 2.10).

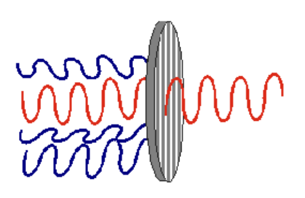


Abbildung 2.10: Linearer Polarisationsfilter [10]

Um mit dieser Technik zwei unterschiedliche Lichtkanäle zu definieren, bedient man sich zweier Projektoren (Abbildung 2.11), deren Lichtkegel überlappend auf eine Leinwand strahlen und in deren Strahlengänge Polarisationsfilter eingebracht werden. Die Polarisationsfilter werden um 90° zueinander verdreht, so dass ein Filter horizontal polarisiertes Licht durchlässt, der andere vertikal polarisiertes Licht. Durch eine Brille mit linearen Polarisationsfilterfolien lassen sich die beiden Stereobildkanäle unmittelbar vor den Augen des Betrachters wieder dekodieren.



Abbildung 2.11: Doppelprojektor mit Polarisationsfiltern [8]

Eine gute Technik, zur Darstellung von Stereobildern auf Röhrenbildschirmen (Computermonitore, Fernsehgeräte) stellen **LC-Shutterbrillen** dar. Die Gläser einer solchen Brille (Abbildung 2.12) enthalten Flüssigkristalle. Im Normalzustand sind sie durchsichtig. Sie verdunkeln sich jedoch durch Anlegen einer elektrischen Spannung. Mittels einer solchen Brille wird jeweils ein Auge des Betrachters, zeitlich versetzt zum anderen Auge, abgedeckt. Röhrenbildschirme haben eine sehr hohe Bildwiederholrate; bei guten Computermonitoren liegt sie bei etwa 160 Hz. Das menschliche Auge nimmt das stark flimmernde Bild, durch die Trägheit der Sehsinneszellen, ab einer Frequenz von ca. 50 Hz als „ruhig“ war. Das bedeutet: Ab einer Bildwiederholrate von 100 Hz (2x50Hz) ist ein zeitlich versetztes Multiplexing von zwei Stereobildkanälen auf einen „Mono“-Bildkanal, also den Röhrenbildschirm, möglich. Die LC-Shutter werden mit dem Stereo-Videosignal synchronisiert und decken jeweils ein Auge des Betrachters ab. Das Auge, für welches das aktuell angezeigte Bild vorgesehen ist bleibt offen. Durch die hohe Bildwiederholrate nimmt der Betrachter kein Flimmern war. Er bemerkt lediglich

eine Verdunklung des Bildes auf etwa die halbe Bildschirmhelligkeit. An diesen Effekt gewöhnen sich die Augen schnell, vergleichbar mit dem Eintritt von einer hellen in eine etwas dunklere Umgebung. Unter Voraussetzung einer guten Abstimmung zwischen Bildschirm und Bildwiederholraten liefern LC-Shutterbrillen eine hinreichende Trennung der beiden Stereobildkanäle. Mit steigender Bildwiederholrate bzw. einer hohen Nachleuchtzeit des Bildschirms steigt die Gefahr des Auftretens sogenannter **Geisterbilder**. Sie beruhen auf mangelnder Trennung der beiden Stereobildteile.



Abbildung 2.12: LCD-Shutterbrille [8]

Die für den 3D Computergrafiksektor vermutlich am besten geeignete Lösung sind die **Virtual Reality Helme** (VR-Helme), manchmal auch unter der Bezeichnung **„Headmounted Displays"** im Handel zu finden (Abbildung 2.13). Ein solches Gerät verfügt über zwei kleine TFT-Bildschirme, die mittels einer optischen Linse direkt vor den Augen des Betrachters platziert werden. Das Stereo-Videosignal wird in der Regel, wie bei der LC-Shuttertechnik, durch ein zeitliches Multiplexing der Einzelbilder erzeugt. Die Elektronik des VR-Helms dekodiert das Signal und verteilt die beiden Teile des Stereobildes auf die TFT-Displays des linken und rechten Auges. Durch die Verwendung zweier Displays fallen die bei LC-Shuttern auftretenden Effekte (Helligkeitsverlust, Geisterbilder) weg. Virtual Reality Helme sind meist sehr teuer (700-1400 EUR) und nur ein einzelner Beobachter kann am 3D-Geschehen Teil haben.

2.3.4 Videosignalaufbereitung

Um stereoskopische Bilddaten an ein Darstellungssystem zu übermitteln muss ein „stereoskopisches Videosignal" aufbereitet werden.

Ein Stereo-Audiosignal dient der Übertragung zweier, getrennter Audiokanäle. Ein **Stereo-Videosignal** überträgt - analog dazu - zwei getrennte Bildkanäle. Die



Abbildung 2.13: i-Glasses Headmounted Displays [11]

Hersteller stereoskopiefähiger PC-Grafikhardware unterstützen im Wesentlichen zwei Verfahren zur Aufbereitung stereoskopischer Videosignale. Diese Verfahren sollen hier, zum besseren Verständnis der weiteren Ausführungen, kurz erläutert werden:

Pageflipping-Verfahren: Das Pageflipping-Verfahren ist im Computerbereich die wohl gängigste Technik zur Aufbereitung stereoskopischer Videosignale. Der Begriff „Pageflipping“ entspringt der Welt der Grafikprogrammierung und beschreibt das schnelle hin- und herschalten („Flipping“) zwischen Bildspeicherbereichen („Pages“) einer Grafikkarte. Zur Generierung eines Stereo-Signals werden im Grafikspeicher zwei unabhängige Bildpuffer angelegt, in denen jeweils ein Kanal des Stereobildes aufgebaut wird (Abbildung 2.14). Die Grafikhardware koppelt abwechselnd, im Takt der vertikalen Bildsynchronisation, einen der beiden Bildpuffer auf ihren Videoausgang aus. Zusätzlich wird das Videosignal mit Synchronisationsinformationen geprägt; sie ermöglichen dem Darstellungsmedium später die Zuordnung der Bildrahmen. Das Pageflipping-Verfahren führt zu einer Verdoppelung der Bildwiederholrate des Videosignals unter Beibehalten der Bildauflösung.

Interlacing-Verfahren: „Interlacing“ beschreibt den Aufbau eines Vollbildes aus zwei, zeitlich versetzt übermittelten, Halbbildern. Dabei repräsentiert

das eine Halbbild alle geraden Bildzeilen, während das zweite Halbbild alle ungeraden Bildzeilen enthält. Um auf diese Weise ein stereoskopisches Videosignal zu erzeugen, wird ein Bildkanal in das erste Halbbild, der zweite Bildkanal in das zweite Halbbild eines Vollbildes geleitet. Abbildung 2.15 demonstriert diesen Vorgang. Die vertikale Auflösung des so entstandenen Stereo-Signals reduziert sich, gegenüber dem gewöhnlichen Videosignal, auf die Hälfte. Die Bildwiederholrate bleibt jedoch dieselbe. Videosignale im PAL-Format übertragen pro Sekunde 25 Vollbilder, bestehend aus 50 Halbbildern. Das Interlacing-Verfahren macht es somit möglich, stereoskopische Videos auf DVD-Medien unter Einhaltung der PAL-Norm zu speichern und auf handelsüblichen Playern wiederzugeben. Die Betrachtung eines solchen Videos kann auf einem gewöhnlichen Fernsehgerät, unter Zuhilfenahme einer LC-Shutterbrille, erfolgen.

Gewöhnlich kann in der Treiber-Konfiguration eines stereoskopiefähigen Grafikadapters zwischen Pageflipping- und Interlacing-Verfahren frei gewählt werden. Das zu wählende Verfahren hängt von Art- und Ausführung der Darstellungsgeräte ab. Geräte, die für den Anschluss an ein Computersystem vorgesehen sind, erwarten in der Regel ein mittels Pageflipping kodiertes Signal.

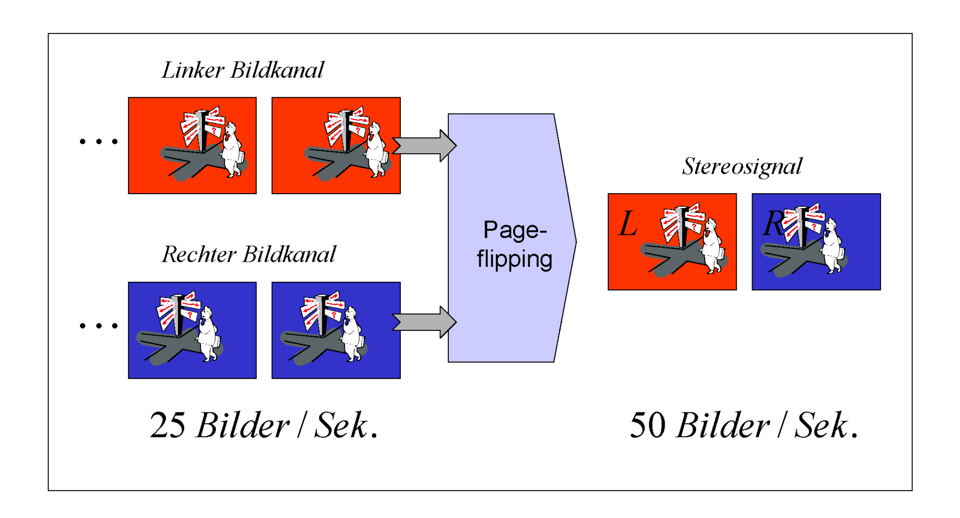


Abbildung 2.14: Pageflipping-Verfahren

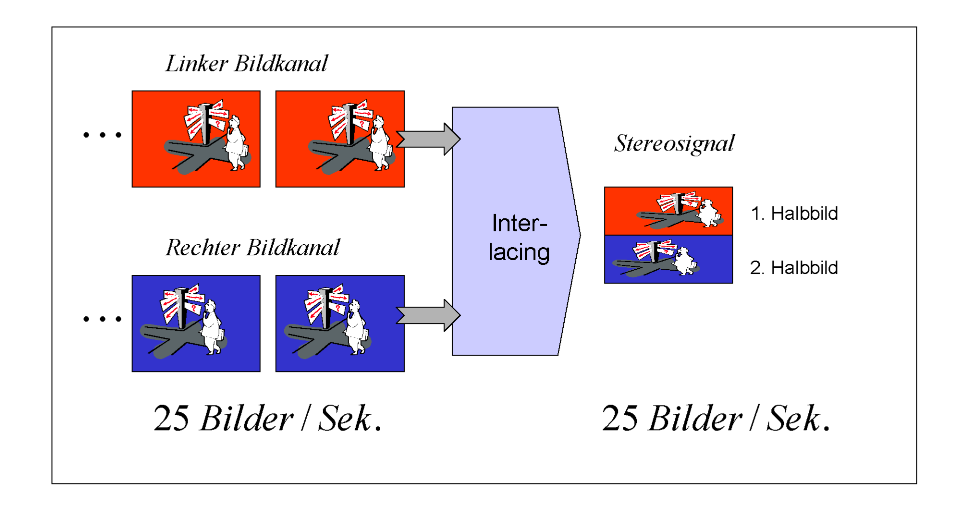


Abbildung 2.15: Interlacing-Verfahren

Kapitel 3

Stereokamera- und Fernsteuersystem

Dieses Kapitel beschreibt einige grundlegende Konzepte, die zur Entwicklung eines Stereokamera- und Fernsteuersystems für den AIBO ERS-7 erarbeitet wurden. Zunächst sollen jedoch Aufgabenstellung und Ziele der Arbeit noch einmal ins Gedächtnis gerufen werden.

3.1 Aufgabenstellung und Ziele der Arbeit

Ziel dieser Arbeit ist es, ein Soft- und Hardwaresystem zu entwickeln, welches den Sony AIBO ERS-7 zu einem Teleroboter erweitert. Dazu soll:

1. Ein **3D-Stereokamerasystem** für den ERS-7 entwickelt werden.
2. Eine Software zur komfortablen **Fernsteuerung** des ERS-7 durch einen menschlichen Bediener realisiert werden.

Die Entwicklung des Stereokamerasystems, welches den ERS-7 um die Fähigkeit des räumlichen Sehens erweitert, steht im Zentrum dieser Arbeit. Dennoch soll der Entwurf einer ergonomischen Schnittstelle zur Fernsteuerung des Roboters nicht vernachlässigt werden.

Am Ende soll ein System stehen, welches dem Menschen ermöglicht in die Welt des Roboters einzutauchen. Und zwar mittels seiner wichtigsten Sinne: Sehen und Hören.

3.2 Fernsteuersystem

Im folgenden soll das Konzept eines Fernsteuersystems für den AIBO ERS-7 Roboter entwickelt werden. Zunächst werden einige allgemeine Betrachtungen zu einer, für den Menschen handhabbaren („ergonomischen") Roboter-Fernsteuerung erörtert. Darauf aufbauend werden Steuerungskomponenten für eine geeignete Benutzerschnittstelle ausgewählt und eine Softwareschnittstelle zur Fernsteuerung des AIBO, das AIBO Remote Framework, vorgestellt.

3.2.1 Ergonomie

Für den Entwurf einer, für den Menschen ergonomischen Fernsteuerung lohnt es sich, die Anatomie und Verhaltensweisen des Menschen in seiner Umwelt näher zu betrachten.

Der Mensch ist ein aufrecht gehendes Wesen, seine Wirbelsäule beschreibt in etwa eine vertikale Achse. Am oberen Ende des Körpers bildet der Hals den Übergang zum Kopf mit den Seh- und Hörsinnesorganen. Der Kopf ist über sieben Halswirbel mit dem Körper verbunden. Muskelstränge entlang der Halswirbel machen den Kopf beweglich. Es ergeben sich, in der Sprache der Robotik ausgedrückt, drei Freiheitsgrade.

1. Der Kopf kann um die Achse der Wirbelsäule gedreht werden.
2. Der Kopf kann in seitlicher Richtung geneigt werden.
3. Der Kopf kann nach vorne und hinten geneigt werden.

Da der Kopf Träger der wichtigen Sinnesorgane (Augen und Ohren) ist, sollte es von Vorteil sein, seine Bewegungen in möglichst „natürlicher" Art und Weise auf

den Kopf des Roboters zu übertragen. Der Roboterkopf sollte also die Bewegungen des menschlichen Kopfes imitieren und nicht über Joystick, Tastatur oder ähnlichem gesteuert werden müssen.

Als nächstes soll das **Verhalten** des Menschen in seiner Umwelt betrachtet werden. Menschen erlernen im Verlaufe ihrer Entwicklung zahlreiche Verhaltensweisen, die ihnen zur Orientierung und Fortbewegung in der Umwelt dienen. Die Imitation einiger einfacher Verhaltensweisen durch die Roboterfernsteuerung vereinfacht die Steuerung des Teleroboters, da dem Bediener überflüssige „Denkarbeit“ abgenommen wird und er sich auf seine eigentliche Aufgabe konzentrieren kann.

Die Verhalten des Menschen können grob in drei Kategorien eingeteilt werden:

1. Reflex-Verhalten.
2. Reaktives Verhalten.
3. Bewusstes, planendes Verhalten.

Reflexe sind angeborene Verhalten. Sie werden durch einen Reiz (Stimulus) oder eine Kombination von Reizen ausgelöst. Bekannte Beispiele sind der Kniereflex oder das Niesen. Ein ausgelöster Reflex führt unmittelbar zu einer Reaktion, ohne Einschaltung des Bewusstseins.

Reaktive Verhalten sind Verhalten, die zwar nicht angeboren sind, aber durch häufiges Einüben so verinnerlicht wurden, dass sie ohne bewusstes Nachdenken ablaufen. Beispiele für solche Verhalten beim Menschen sind das aufrechte Gehen, das Verfolgen sich bewegender Objekte oder auch das Drehen des Kopfes in Richtung eines lauten Geräusches.

Die dritte Kategorie, das **bewusste, planende Verhalten,** ist zur Simulation durch eine Robotersteuerung ungeeignet. Vielmehr ist das planende Verhalten gerade das Verhalten, auf welches sich der Bediener eines Teleroboters konzentrieren können sollte. Die beiden anderen Kategorien, Reflexe und reaktives Verhalten, dürfen jedoch für die Entwicklung einer guten Steuerung näher betrachtet werden.

3.2.2 Steuerungskomponenten

Zur Steuerung eines zum Teleroboter umgebauten AIBO ERS-7 muss, neben der Steuersoftware, ein geeignetes hardwareseitiges Benutzer-Interface erstellt werden. Die Standardeingabegeräte eines handelsüblichen PCs, Maus und Tastatur, reichen nicht aus.

Zur Auswahl der neuen Eingabegeräte soll zunächst erörtert werden, welche Funktionalitäten des AIBOs ferngesteuert werden sollen:

1. Der AIBO, als mobiler Roboter, soll sich in seiner Umgebung, soweit es seine Konstruktion zulässt, frei bewegen können.
2. Der Kopf des AIBO trägt später die Stereokamera und die Stereomikrofone. Aus diesem Grunde muss der Kopf beweglich sein, damit der Benutzer Blick- und Hörrichtung frei variieren kann.
3. Der AIBO soll später einige vordefinierte (eingeübte) Verhalten zeigen können. Das sind vor allem vordefinierte Bewegungsabläufe wie eine automatische 180° Wende, Begrüßungsbellen, oder das Aufführen eines Tanzes.

Zu 1: Der AIBO ist ein „vierbeiniger“ Roboterhund. Die Koordination der Gelenke für einen sicheren Gang ist sehr komplex und muss daher von der Robotersteuerung automatisiert werden. Diese Aufgabe übernimmt bereits das Betriebssystem des ERS-7. Durch die Steuerungsautomatisierung reduziert sich die Bewegungssteuerung des ERS-7 auf einige wenige Funktionen.

Laufen: Vorwärts/rückwärts.

Drehen: Linksdrehung, Rechtsdrehung auf der Stelle.

Bogenlauf: Vorwärts-/Rückwärtslauf auf einem Kreisbogen unter Angabe eines Winkels.

All diese Bewegungen kann der AIBO in den drei Geschwindigkeitsstufen „langsam“, „normal“ und „schnell“ durchführen.

Als Eingabegerät zur Steuerung der Laufbewegungen genügt, dank der Vereinfachungen durch die interne Software des AIBO, ein handelsüblicher Steuerknüppel

(Joystick) mit zwei analogen Achsen (x- und y-Achse). Die geplante Umsetzung der Joystickeingaben durch den AIBO ERS-7 zeigt Tabelle 3.1 und Abbildung 3.1.

Tabelle 3.1: Umsetzung der Joystickeingaben durch den AIBO ERS-7

Steuerknüppeleingabe	Umsetzung durch AIBO
nach vorne	läuft vorwärts
nach hinten	läuft rückwärts
nach links	dreht links herum
nach rechts	dreht rechts herum
nach vorne, links	Bogenlauf, Linksbogen
nach vorne, rechts	Bogenlauf, Rechtsbogen

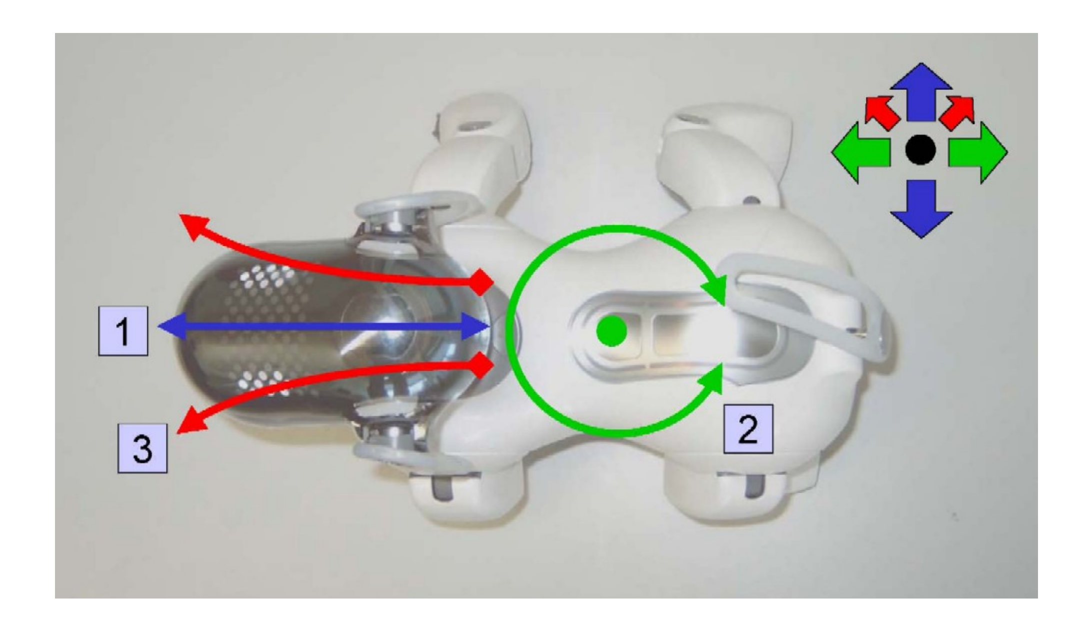


Abbildung 3.1: Umsetzung der Joystickeingaben durch den AIBO ERS-7

Tabelle 3.2 zeigt die Übertragung analoger Joystickachsen auf diskrete Werte. Aus diesen Werten können Steuerkommandos für den Roboter hergeleitet werden.

Tabelle 3.3 zeigt einige Beispiele, um die Generierung der Steuerkommandos zu verdeutlichen:

Zu 2: Es wurde bereits darauf hingewiesen, dass der Kopf des AIBO das Stereokamerasystem und die Stereomikrofone trägt. Eine Kopfsteuerung per Joystick

Tabelle 3.2: Abbildung der analogen Joystickachsen auf diskreten Werten

Funktion	Achse(n)	X	Y
Laufen: Vor/rück	y	0	[-3, -2, -1, 0, 1, 2, 3]
Drehen: Links/rechts	x	[-3, -2, -1, 0, 1, 2, 3]	0
Bogenlauf	x, y	[-60°, ..., 0°, ..., 60°]	[-3, -2, -1, 0, 1, 2, 3]

Tabelle 3.3: Beispiele zur Generierung von Steuerkommandos

Achsstellungen	Steuerkommando
x=0, y=3	Vorwärtslauf, Geschwindigkeitsstufe 3 (schnell)
x=(-1), y=0	Linksdrehung (-N), Geschwindigkeitsstufe 1 (langsam)
x=0, y=0	Stop
x=1, y=2	Rechtskurve , Winkelstufe 1 (+20°), Geschw. Stufe 2 (mittel)

wäre „unergonomisch“, da der Mensch seine Blick- und Hörrichtung gewöhnlich durch das Drehen und Neigen seines Kopfes ändert. Um eine natürliche Steuerung des Roboterkopfes durch den Bediener zu ermöglichen wird ein sogenannter **Headtracker** eingesetzt. Dieses Eingabegerät besteht aus einer Infrarotkamera, die einen **IR-Reflexionspunkt** verfolgt. Dessen Position wird in absolute Koordinaten (im Bereich des IR-Kamerasichtfeldes) umgewandelt. Bewegliche Objekte, also auch der Kopf des Bedieners, können durch einen IR-Reflektionspunkt markiert und verfolgt werden. Abbildung 3.2 illustriert die Funktionsweise des verwendeten Headtrackers „Track IR“ der Firma Natural Point.

Der Kopf des AIBO ERS-7 verfügt über drei Freiheitsgrade (Abbildung 3.3). Die Drehung des Kopfes um die Halsachse (#1), die Neigung des Kopfes durch das Kopfgelenk (#2) und die Neigung von Hals und Kopf durch das Halsgelenk (#3). Das Betriebssystem des ERS-7 koordiniert Hals- und Kopfgelenk automatisch, abhängig von der Haltung des Roboters (liegend, sitzend, stehend). Zur Ausrichtung des AIBO Kopfes genügt somit die Angabe eines horizontalen und vertikalen „Kopfwinkels“ (engl. Headangle). Der Kopf kann aus der Mittelstellung um bis zu 93° nach links oder rechts gedreht werden, um bis zu 50° nach oben geneigt und um bis zu 20° nach unten geneigt werden. Die Haltung des Robo-

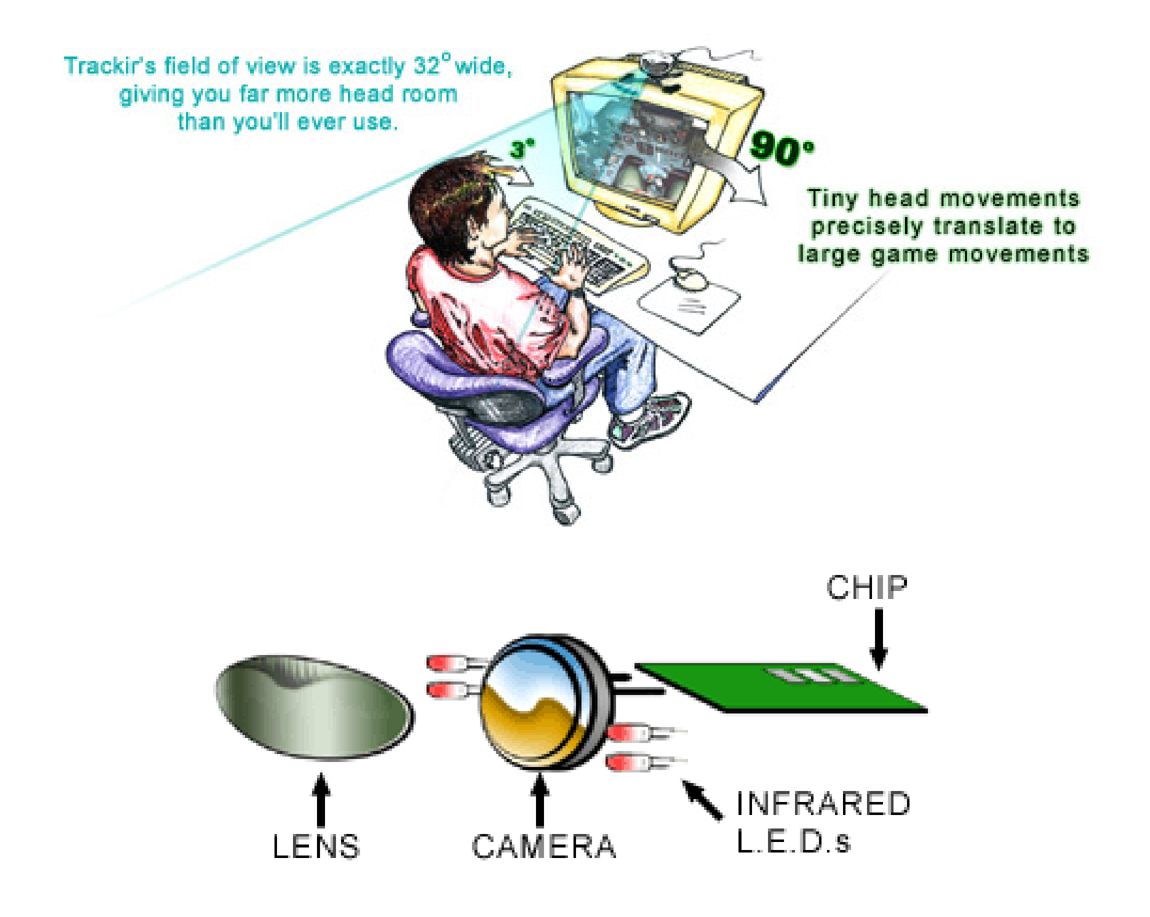


Abbildung 3.2: Headtracker „Track IR“ der Fa. Natural Point [16]

ters beeinflusst - dank automatischer Ausgleichsfunktionen - die Blickrichtung des Roboters nicht.

Die Anatomie des Roboters unterscheidet sich, in der Beweglichkeit des Kopfes, von der des Menschen. Ihm fehlt die Möglichkeit den Kopf nach links oder rechts zu neigen. Diese Begebenheit stellt keinen besonderen Nachteil dar, da der Mensch seinen Kopf gewöhnlich selten zur Seite neigt. Er tut dies hauptsächlich um eine seitliche Neigung seines Körpers auszugleichen und den Kopf in einer senkrechten Position zu halten.

Der Headtracker liefert, bei Verfolgung des Reflektionspunktes am Kopf des Bedieners, Positionsdaten. Diese können ausgewertet und so genutzt werden, dass der AIBO ERS-7 die Kopfbewegungen seines Bedieners imitiert.

Es gibt jedoch noch eine weitere anatomische Besonderheit des Roboters. AIBO besitzt im Wesentlichen die Anatomie eines Vierbeiners. Die Halswirbelsäule

eines Menschen hat eine nahezu senkrechte Ausrichtung, wodurch der Kopf in horizontaler Ebene gedreht werden kann. Der Hals des AIBO dagegen ist um ca. 35° nach vorne geneigt; Abbildung 3.4 verdeutlicht dies.

Die Folge ist: Der Roboter dreht seinen Kopf bei Rotation um die Halsachse nicht nach links oder rechts, sondern schaut nach links/oben bzw. rechts/oben. Den Sachverhalt kann man sich durch einen einfachen Selbstversuch verdeutlichen, indem man den Oberkörper um 90° nach unten beugt, den Blick nach vorne richtet und den Kopf nach links oder rechts dreht.

Der Bediener wird also durch die Nutzung des Fernsteuer- und Kamerasystems den Eindruck bekommen, sich im Körper eines Hundes zu befinden.

zu 3: Eine letzte Art von Steuersignalen, die der Bediener dem AIBO übermitteln können muss, sind Signale, die vordefinierte Verhaltensmuster anstoßen. Da ein solches Verhaltensmuster nicht parametrisiert wird, reicht dazu eine binäre Eingabe. Es können entweder Tasten der Computertastatur belegt werden oder die Knöpfe des Steuerknüppels genutzt werden. Da moderne Joysticks eine große Zahl frei belegbarer, digitaler Knöpfe bieten, scheint diese Möglichkeit sinnvoll. Durch das Drücken einer der Joysticktasten können dann Bewegungsabläufe (180° Wende, Begrüßungsbellen, oder Tanzen) ausgelöst werden.

3.2.3 Zustandsrückmeldungen

Der AIBO ERS-7 verfügt über eine ganze Reihe eingebauter Sensoren. Die Anzeige ausgewählter Sensordaten könnte die Bedienung des Roboters erleichtern.

Es ist sinnvoll dem Bediener einige Informationen über den inneren Zustand des Roboters zu übermitteln. Dazu gehören der Ladezustand der internen Batterie zur Vermeidung eines unerwarteten Ausfalls des Systems, aber auch Informationen über den aktuellen Betriebszustand des Roboters. Wenn AIBO beispielsweise gerade einen vordefinierten Bewegungsablauf ausführt, kann er nicht auf Eingaben des Steuerknüppels reagieren. Also wäre es möglicherweise interessant für den Bediener zu wissen, wann der Bewegungsablauf abgeschlossen ist.

Des weiteren verfügt AIBO über zwei Infrarot-Entfernungsmesssensoren. Die von ihnen gelieferten Daten erleichtern, richtig aufbereitet, dem Bediener die Navigation mit dem Roboter. Von den beiden Entfernungsmesssensoren befindet sich

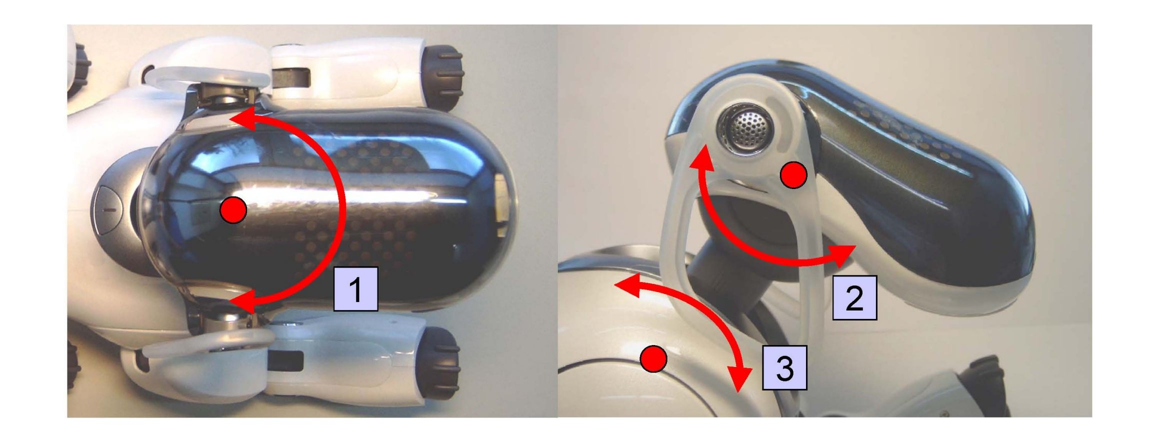


Abbildung 3.3: Freiheitsgrade des AIBO-Kopfes

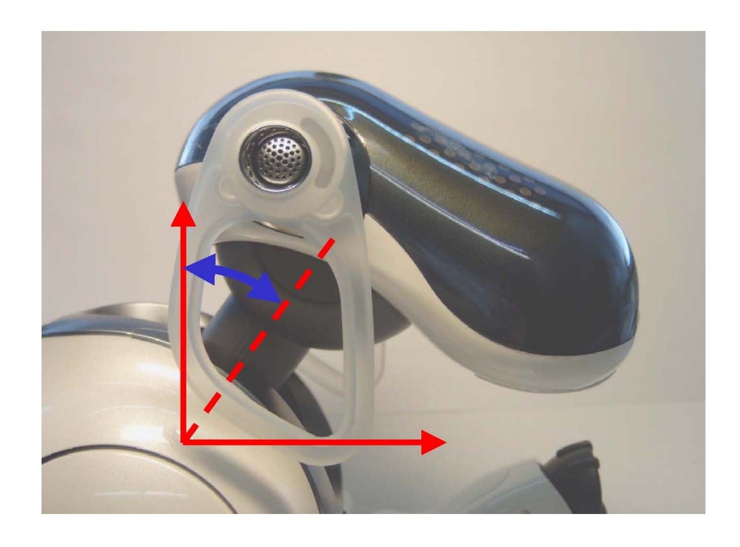


Abbildung 3.4: Neigung des AIBO-Halses

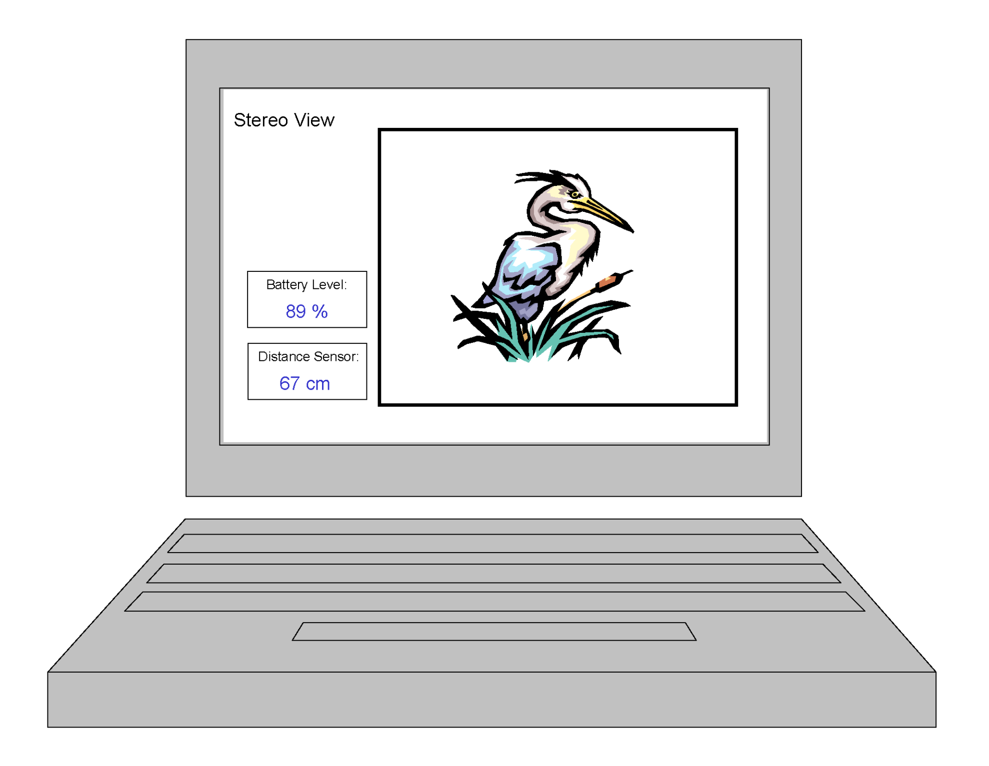


Abbildung 3.5: Schema; Benutzerschnittstelle

einer im Kopf des ERS-7. Er misst die Entfernung zu dem Objekt, das sich in der Bildmitte der Kamera befindet. Die Anzeige der Daten dieses Sensors erleichtert dem Bediener die Abschätzung von Entfernungen zu Objekten in seiner Blickrichtung. Der zweite Entfernungsmesssensor sitzt in der Brust des Roboters und ist um einen Winkel von 60° nach unten geneigt. Eine automatische Auswertung der Daten dieses Sensors ermöglicht der Robotersteuerung das Erkennen von Abgründen (z.b. Treppenabsätzen). Eine Anzeige seiner Daten auf dem Stereodisplay ist aber nicht sinnvoll.

Sensorinformationen, die dem Bediener den Umgang mit dem Roboter erleichtern (Batteriezustand, Entfernung zum Objekt in der Bildmitte), sollten jedoch in der grafischen Benutzerschnittstelle der Fernsteuerungssoftware angezeigt werden.

Abbildung 3.5 zeigt eine schematische Darstellung der geplanten Benutzerschnittstelle. Am linken Bildrand werden die wichtigsten Informationen und Daten des Roboters eingeblendet. Der weitaus größte Bildbereich wird zur Anzeige des Stereobildes verwendet.

3.2.4 AIBO Remote Framework

Die Firma Sony bietet für den Roboter AIBO ERS-7 einige kostenlose Softwareentwicklungspakete zum freien Download an. Eines davon ist das **AIBO Remote Framework SDK**.

Das AIBO Remote Framework dient zur Entwicklung von Software für Windows PC Plattformen unter Verwendung des Microsoft „Visual C++“-Compilers. Eine Remote Framework Anwendung kommuniziert über Wireless-LAN mit dem AIBO und ist in der Lage seine Funktionen von einem lokalen PC fernzusteuern.

Das im Juli 2004 von Sony herausgegebene AIBO Remote Framework ist eine gute Lösung zur Entwicklung eines ergonomischen Fernsteuersystems. Leider befindet es sich zum Zeitpunkt der Erstellung dieser Diplomarbeit noch in einem „Beta“-Stadium. Neben möglichen Fehlern in der Software erschwert eine unzureichende Dokumentation[1] den Einsatz des AIBO Remote Frameworks.

3.3 Stereokamerasystem

In den folgenden Abschnitten wird erörtert, welche Vorteile der Einsatz eines Stereokamerasystems auf dem AIBO Roboter bietet. Außerdem wird dargelegt, welche Anforderungen ein solches System an die einzusetzenden Hard- und Softwarekomponenten stellt.

3.3.1 Warum Stereokamera

Der AIBO ERS-7 verfügt von Hause aus über einen, in die „Nase“ des Roboters eingebauten, **CMOS-Bildsensor.** Bilder dieser internen Kamera werden durch die Software des Roboters verarbeitet und über die Wireless-LAN Schnittstelle versendet. Trotz des mit großzügigen 350.000 Bildpunkten angegebenen Sensors ist die zu erzielende Bildqualität und Bildrate - vor allem bei der Übertragung bewegter Bilder - sehr gering. Auch die geringe Lichtempfindlichkeit des Bildsensors schafft viele Probleme. Eine stereoskopische Bildaufnahme ist mit der Grundausstattung des AIBO ERS-7 natürlich ebenfalls nicht möglich. Das heißt,

[1] Diese ist praktisch nur in Form wenig kommentierter Beispielprogramme vorhanden.

der Bediener eines AIBO Fernsteuersystems kann über den Roboter nicht räumlich sehen und die damit verbundenen Vorteile nicht nutzen.

Die hier angestrebte Entwicklung eines Stereokamerasystems für den AIBO ERS-7 wird demnach Nachteile des Roboters (u.a. die schlechte Bildqualität) ausgleichen und interessante neue Möglichkeiten des Sehens eröffnen.

Das Stereokamerasystem ist grundsätzlich vom Trägerroboter unabhängig und wird nicht zwingend auf dem AIBO ERS-7 eingesetzt werden müssen. Sehr interessant ist möglicherweise der Einsatz des Systems auf einem Teleroboter mit Greifarmen. Mittels eines solchen Roboters könnten die Vorteile des räumlichen Sehens ausgeschöpft werden. Mit den Greifern könnten, von einem Menschen ferngesteuert oder sogar durch die Robotersteuerung selbst, präzise Arbeiten verrichtet werden.

3.3.2 Hardware

Für die Auswahl der passenden Hardware, zum Aufbau eines Stereokamerasystems für den ERS-7, gibt es zwei wesentliche Parameter: Erstens die geringe Größe des Roboters und zweitens die freie Mobilität des AIBO ERS-7. Die geringen Abmessungen des Roboters, vor allem des Kopfes, der als Träger für das Kamerasystem dienen soll, erfordern sehr kleine und leichte Kameras. Der zweite Parameter, die Mobilität, erfordert einen drahtlosen Übertragungsmechanismus für die, von den Kameras gelieferten, Bilder.

Um den genannten Vorgaben gerecht zu werden, bietet sich die Konstruktion eines **Mini-Doppelkamerasystems** an. Als Kameras werden zwei, normalerweise in der Überwachungstechnik eingesetzte, **Mini-Funkkameras** des Typs **ZT-811** verwendet. Die 20x20 mm großen, ca. 20g schweren Kameras sind intern mit einem analogen 2,4 GHz Funksender ausgestattet und werden in der Regel als Set mit dem passenden analogen Empfangsmodul ausgeliefert. Die ZT-811 Funkkameras liefern mit einer CMOS-Chip Auflösung von 628x582 Pixel, sowie hoher Lichtempfindlichkeit und Bildwiederholrate eine gute Bildqualität.

Zur Montage und Ausrichtung der Kameras auf dem Kopf des AIBO ERS-7 ist die Konstruktion einer speziellen **Kamerahalterung** erforderlich. Die Halterung muss zwei ZT-811 Kameras in einem Horizontalabstand von 6 cm (Augenabstand)

aufnehmen. Die Kameras müssen, zur Ausrichtung, einzeln in horizontaler und vertikaler Ebene eingestellt werden können.

Bevor die Signale der beiden Kameras durch die PC-Software verarbeitet werden können, müssen die analogen Bilddaten digitalisiert werden. Das **Capturing** (dt. Einfangen) von zwei Videosignalen auf einen PC gestaltet sich schwierig, da die üblicherweise im Handel erhältlichen **Videocapturegeräte**[2] nicht kaskadierbar sind. Es kann aus treibertechnischen Gründen immer nur eine Karte in denselben Rechner eingesetzt werden. Auch USB-Videocapturegeräte schaffen hier keine Abhilfe. Die einzige Möglichkeit dieses Problem angemessen zu lösen, ist der Einsatz spezieller Profihardware für sogenannte Video-Streaming-Server[3]. Die amerikanische Firma „Viewcast" ist einer der wenigen Anbieter solcher Spezialgeräte. Zur Erstellung einer Notebook kompatiblen Lösung, viel die Wahl auf zwei kaskadierbare **USB-Videocaptureboxen** des Typs **Osprey-50** der Firma Viewcast. Eine Alternative wäre eine **Multikanal Videocapture PCI-Karte**[4] zum Einsatz in einem Desktop PC. Bei dieser Alternativlösung ist darauf zu achten, dass eine Karte ausgesucht wird, die mehrere Capturechipsätze besitzt. Karten, die lediglich mehrere Videoeingänge nacheinander auf ein und denselben Capturechipsatz schalten sind ungeeignet.

Zur Darstellung der, durch die 3D-Software verarbeiteten, stereoskopischen Bildinformationen ist ein stereoskopiefähiger Grafikadapter erforderlich. Zurzeit erfüllen Grafikadapter des Chipherstellers „NVIDIA" diese Bedingung am zufriedenstellendsten, so dass für diese Arbeit eine **NVIDIA Grafikkarte** mit einem **„GeForce 4"-Chipsatz** Verwendung fand. Stereoskopiefähige Grafikkarten anderer Hersteller sind theoretisch auch einzusetzen, da bei der Entwicklung der Software auf eine Spezialisierung auf NVIDIA Grafikchipsätze verzichtet wurde.

Als **stereoskopisches Darstellungsmedium** wird im Rahmen dieser Arbeit ein **„head mounted Virtual Reality Display"** verwendet, das i-Glasses der Firma IO-Displays (Abbildung 2.13). Es ist aber auch möglich andere stereoskopische Darstellungssysteme[5] zu verwenden, beispielsweise eine LC-Shutterbrille.

[2]Häufig in Form von PC-Erweiterungskarten oder externen USB-Geräten erhältlich.

[3]Video-Streaming-Server: Netzwerkserver zum Sammeln und Verteilein von Videoinformationen, beispielsweise für das Internetfernsehen.

[4]Videocapturekarte mit mehreren parallel nutzbaren analogen Videoeingängen.

[5]Sofern diese mit der verwendeten Grafikkarte kompatibel sind.

3.3.3 Software

Die für das Stereokamerasystem zu entwickelnde Computersoftware hat die Aufgabe, die Videostreams der beiden Stereo-Bildkanäle nach ihrer Digitalisierung entgegenzunehmen, zu verarbeiten und korrekt über einen stereoskopiefähigen Grafikadapter darzustellen.

Da die Software für die Windows Plattformen entworfen werden soll, bietet sich zur Programmierung die integrierte Entwicklungsumgebung **„Microsoft Visual Studio"** in Kombination mit dem **„Microsoft Visual C++"-Compiler** an. Zur Verarbeitung von Multimediadaten eignet sich die Multimediaschnittstelle **„Direct X"** in der Version 9. Sie wird von der Firma Microsoft kostenlos angeboten.

Direct X 9 ermöglicht die Ansteuerung unterschiedlicher Klassen von Videoeingabegeräten durch die Komponente **„Directshow"** und die Darstellung von 3D-Computergrafik durch die Komponente **„Direct-3D"**. Leider existiert in der Version 9 der Direct X Softwareschnittstelle keine Funktionalität zur Darstellung stereoskopischer Bilddaten. Dieses Problem lässt sich, wie später gezeigt wird, auf unterschiedlichen Wegen lösen.

Zur Darstellung stereoskopischer Bilddaten liefert der Grafikchiphersteller NVIDIA ein spezielles Entwicklungspaket, das **„NVIDIA Stereo Blit SDK"**. Die Bezeichnung SDK (Software Development Kit) ist stark übertrieben. Es handelt sich dabei lediglich um einen Beispielsourcecode, der von NVIDIA nicht offiziell freigegeben wurde und darüber hinaus mit keiner getesteten NVIDIA Grafikkarte funktionierte. In Anbetracht der Systemstabilität und Kompatibilität zu anderen Grafikchipherstellern wurde daher auf die Verwendung des NVIDIA Stereo Blit SDKs verzichtet. Dennoch soll diese Möglichkeit hier erwähnt werden um zukünftigen, hoffentlich brauchbareren Versionen dieses SDKs Rechnung zu tragen.

Kapitel 4

Technische Realisierung

In diesem Kapitel wird zunächst ein Überblick über das Gesamtsystem vermittelt und darauf aufbauend die technische Realisierung des Fernsteuersystems, sowie des Stereokamerasystems erläutert.

4.1 Überblick: Gesamtsystem

Die folgenden Abschnitte vermitteln einen Überblick über die Hard- und Softwarearchitektur des Stereokamera- und Fernsteuersystems und zeigen die Kommunikationswege zwischen den einzelnen Komponenten.

4.1.1 Hardwarearchitektur

Abbildung 4.1 zeigt die Hardwarearchitektur des entwickelten Stereokamera- und Fernsteuersystems mit zugehörigen externen Geräten.

Die Abbildung gliedert sich in zwei Teile. Der obere Teil zeigt das **„lokale System“** von dem der AIBO Roboter ferngesteuert wird.

Die **Benutzerschnittstelle** des Systems besteht aus vier Geräten:

Joystick: Steuert die Laufbewegungen des AIBO ERS-7.

Headtracker: Steuert die Ausrichtung des Roboterkopfes und der Stereokamera.

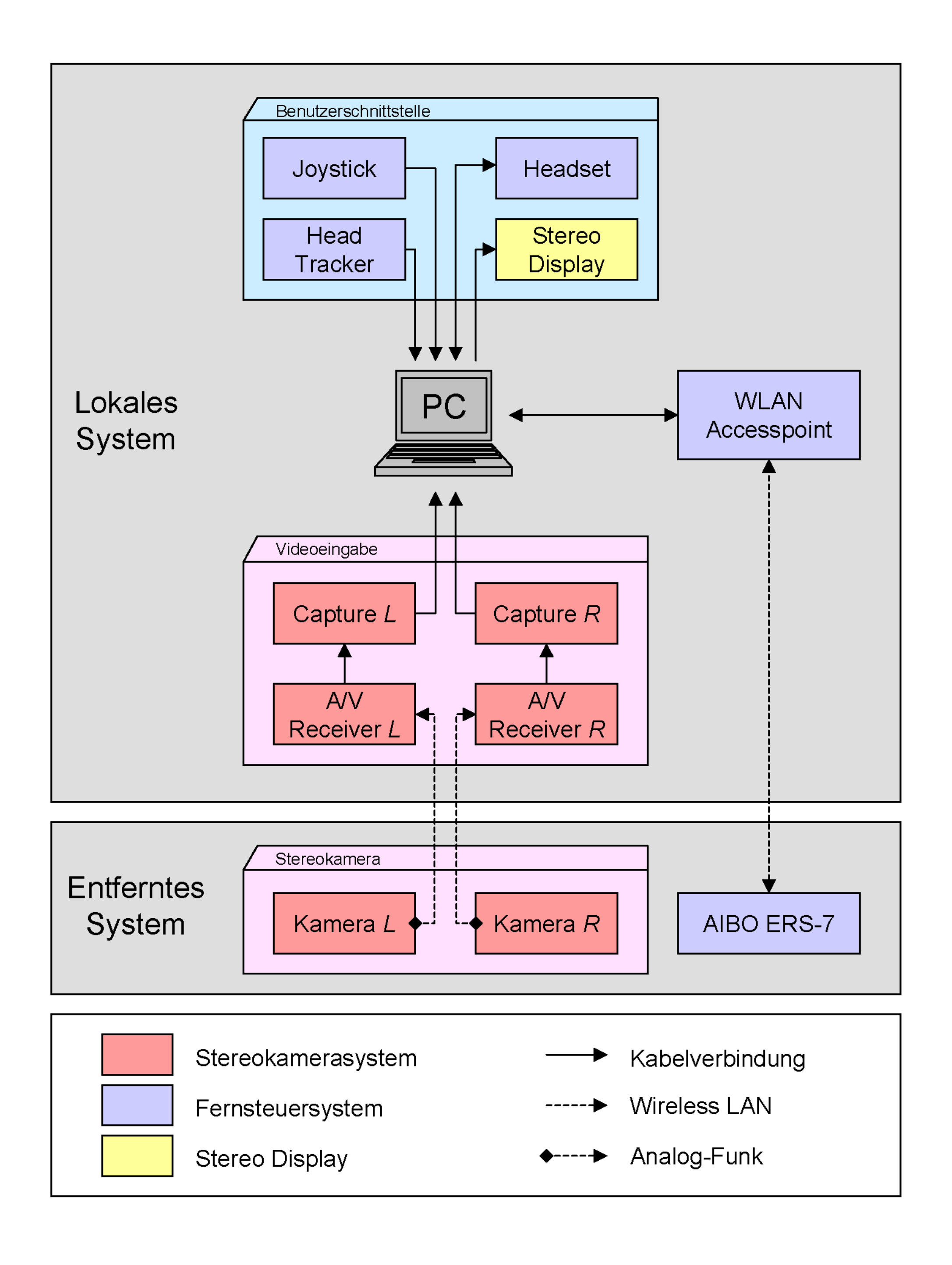


Abbildung 4.1: Hardwarearchitektur des Gesamtsystems

Headset: Ermöglicht die akustische Kommunikation (Hören und Sprechen) über den Roboter.

Stereo Display: Ermöglicht die stereoskopische Darstellung der Kamerabilder.

Joystick und Headtracker werden an die USB-Schnittstellen des Steuer-PCs angeschlossen. Das Headset wird mit den Audio Ein- und Ausgängen der Soundkarte verbunden. Als Stereo Display wurde in dieser Arbeit ein i-Glasses Virtual Reality Display verwendet. Dieses wird an dem Monitorausgang der Grafikkarte angeschlossen.

Der „lokale“ Teil des **Stereokamerasystems** besteht aus zwei 2,4 GHz A/V Receivern („A/V Receiver L“ und „A/V Receiver R“). Die Geräte empfangen die Funksignale der Stereokamera und leiten die analogen Kamerabilder über Videokabel an die zwei Videocapturegeräte („Capture L“ und „Capture R“). Diese digitalisieren die Videosignale und leiten sie über zwei USB-Schnittstellen in den Steuer-PC.

Der untere Teil der Abbildung zeigt das **„entfernte System“**, den AIBO Roboter mit der aufgesetzten Stereokamera. Der AIBO ERS-7 ist über seine W-LAN-Schnittstelle mit dem Steuer-PC verbunden. Die Stereokamera „funkt“ die aufgenommenen Bilder auf dem analogen 2,4 GHz Band zu den A/V Receivern des „lokalen Systems“.

4.1.2 Softwarearchitektur

Abbildung 4.2 zeigt die Architektur der, für das Stereokamera- und Fernsteuersystem entwickelten Software. Die Software wird im folgenden als „Stereoview-Anwendung“ bezeichnet.

Die Anwendung gliedert sich in zwei Programmmodule: Das „AIBO Fernsteuersystem“ und das „Stereokamerasystem“. Zwischen den Modulen existiert im Wesentlichen nur eine Schnittstelle; über diese werden ausgewählte Sensordaten des AIBO ERS-7 (Batteriezustand, Entfernungssensor) in die stereoskopische Bildausgabe eingebunden.

Das Software des Fernsteuersystems baut auf dem AIBO Remote Framework auf. Das Framework repräsentiert einen realen AIBO ERS-7 auf dem fernsteuernden

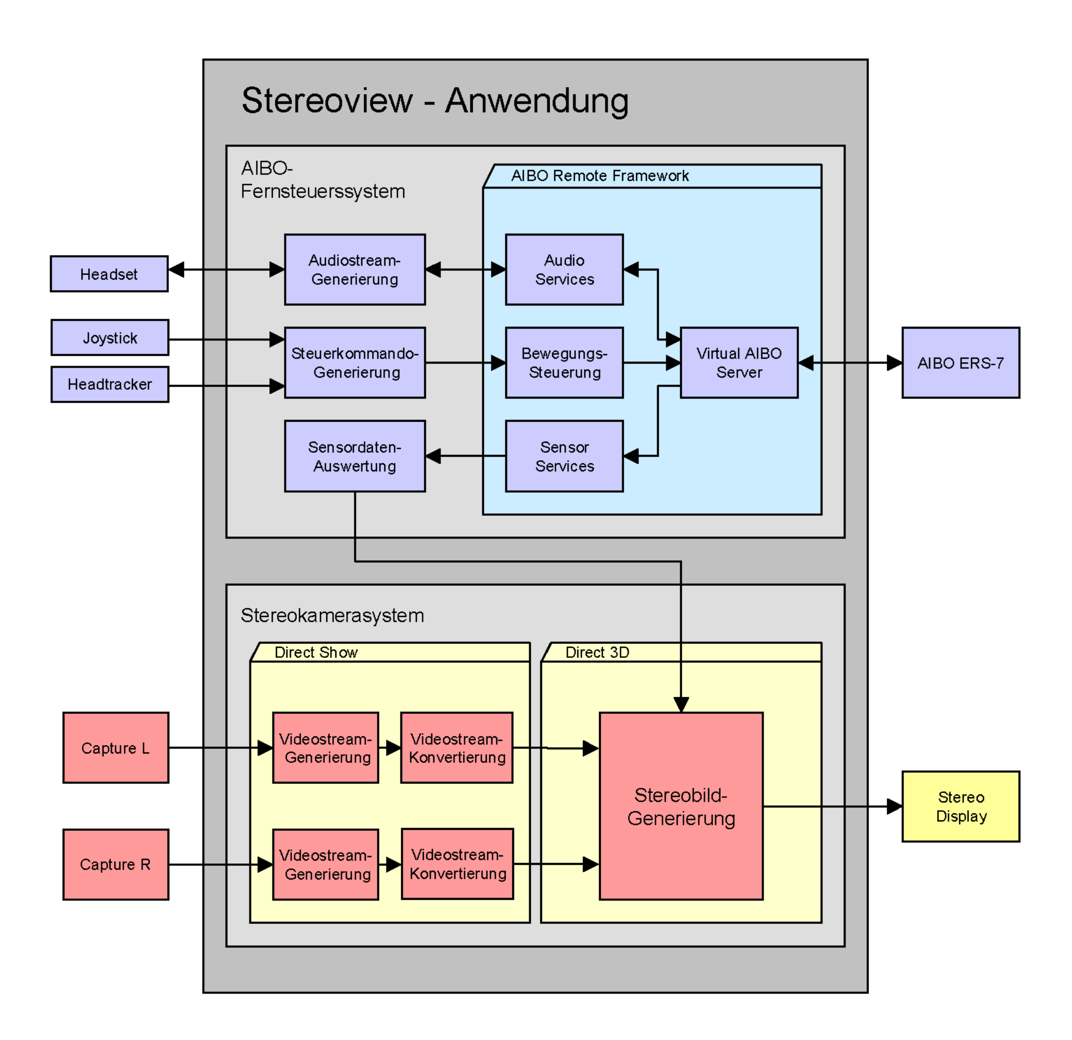


Abbildung 4.2: Softwarearchitektur der Stereoview-Anwendung

System (Steuer-PC) als ein Objekt der Klasse „Virtual AIBO“. Über dieses Objekt können die Funktionen des Roboters gesteuert werden und Zustände abgefragt werden.

Die Softwaremodule Fernsteuerung und Stereokamera sind unabhängig voneinander einsetzbar.

4.2 Entwicklungsumgebung

Die technische Realisierung des Stereokamera- und Fernsteuersystems für den AIBO ERS-7 geschieht auf Grundlage einer, aus verschiedenen Entwicklungspaketen zusammengesetzten, Entwicklungsumgebung. Im folgenden werden die einzelnen Komponenten der Entwicklungsumgebung und deren Installation beschrieben.

4.2.1 Microsoft Visual C++ 6 und Direct X 9 SDK

Der **„Microsoft Visual C++“-Compiler**, zusammen mit der zugehörigen Entwicklungsumgebung, dem „Visual Studio“, eignet sich hervorragend zur Erstellung von Computerprogrammen für die Microsoft Windows Plattformen. Durch das Hinzufügen des **Direct X 9 Software Development Kits** wird das Entwicklungspaket um eine sehr leistungsfähige Softwareschnittstelle zur Entwicklung von Multimediaapplikationen erweitert.

4.2.1.1 Installation

Zur Installation des „Visual C++ 6“-Compilers folgen sie, nach Einlegen der Installations-CD den Anweisungen auf dem Bildschirm. Nach erfolgreicher Installation des Compilers wird das Direct X 9 SDK[1] nachinstalliert. Beim Aufspielen des Direct X SDKs ist unbedingt darauf zu achten, dass die Debug Version installiert wird. Diese Version ermöglicht später ein einfacheres und gezielteres Debuggen neu entwickelter Software durch den im Visual Studio enthaltenen Debugger.

[1] Kostenlos erhältlich unter www.microsoft.com.

Nach der Installation des Direct X SDKs müssen die Einstellungen im Visual Studio geprüft und eventuell angepasst werden, um die korrekte Einbindung des SDKs zu garantieren. Wählen sie dazu im Menüpunkt <Extras> der Entwicklungsumgebung den Punkt <Optionen>. Überprüfen sie dort, unter der Registerkarte <Verzeichnisse>, ob die *INCLUDE*- und *LIB*-Verzeichnisse des Direct X 9 SDKs korrekt in der Entwicklungsumgebung eingebunden sind. Die Verzeichnisse sollten in der Liste über den compilereigenen Verzeichnissen erscheinen. Damit wird vermieden, dass der Compiler die mitgelieferten, gleichnamigen und in der Regel veralteten Direct X *INCLUDE*- und *LIB*-Dateien verwendet.

Entsprechen die Einstellungen der obigen Beschreibung, kann mit der Einrichtung eines Projekts fortgefahren werden.

4.2.1.2 Einrichtung eines Projekts

Die Einrichtung eines „Visual C++"-Projekts mit Unterstützung des Direct X 9 SDKs stellt für „Neulinge" häufig eine Hürde dar. Aus diesem Grunde möchte ich diesen Vorgang hier kurz umreißen und dem Leser einige Hinweise geben.

Zur Erstellung eines neuen Direct X 9 Projekts, öffnen sie den „Visual C++"-Compiler und wählen sie im Menü <Datei> den Unterpunkt <Neu>. Wählen sie in der erscheinenden Dialogbox in der Registerkarte <Projekte> den Listeneintrag „Win32 Anwendung" und legen sie ein neues Projekt an.

Um ein Projekt mit Direct X 9 Unterstützung erfolgreich compilieren und binden („linken") zu können, ist es notwendig der Entwicklungsumgebung mitzuteilen, welche Compilerbibliotheken (Libraries) in den Linkervorgang einbezogen werden sollen.

Dazu gehen sie folgendermaßen vor: Wählen sie im Menüpunkt <Projekt> den Unterpunkt <Einstellungen>. Im Projekteinstellungsdialog wählen sie im linken, oberen Kombinationsfeld „WIN32 DEBUG" aus, um die Einstellungen für die Debug-Konfiguration des Projekts anzupassen. Aktivieren sie die Registerkarte <Linker> und setzten sie den Schreibcursor in das Eingabefeld <Objekt-/Bibliothek-Module>. Dieses Eingabefeld muss um alle zu verwendenden Bibliotheken ergänzt werden (Abbildung 4.3).

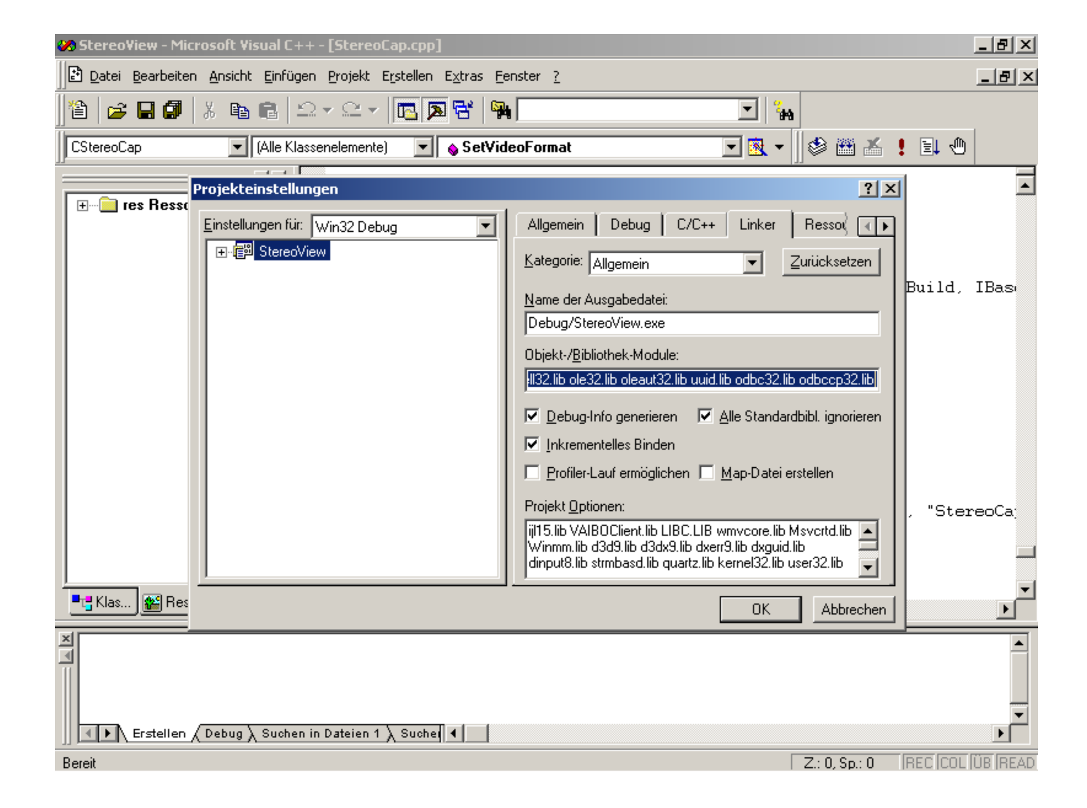


Abbildung 4.3: Visual C++ Projekteinstellungen

Ergänzen sie zur Erstellung eines Projekts, welches Direct-3D, Directshow und Directsound einsetzt folgende zusätzlichen Bibliothekseinträge:

„winmm.lib d3d9.lib dsound9.lib d3dx9.lib dxerr9.lib dxguid.lib strmbasd.lib quartz.lib“.

Es wurde in den vorhergehenden Abschnitten beispielhaft gezeigt, wie der „Microsoft Visual C++“-Compiler für die Verwendung mit dem Direct X 9 SDK einzurichten ist. Zum Erwerb fundierten Wissens ist jedoch das Studium von Fachliteratur (z.b. [2]) oder der mit dem Direct X 9 SDK gelieferten Online-Dokumentation zu empfehlen.

4.2.2 AIBO Remote Framework SDK

Das **AIBO Remote Framework SDK** ist ein Entwicklungspaket zur Erstellung von Anwendungen, die es ermöglichen den AIBO ERS-7 über einen Windows PC fernzusteuern. Durch die Verwendung des AIBO Remote Frameworks wird

der ERS-7 zu einem teilautonomen Teleroboter. In den folgenden Abschnitten wird grob skizziert, wie das Framework auf dem ERS-7 und dem PC installiert wird. Die Komponenten des Frameworks können kostenlos von den AIBO Entwicklerseiten der Firma Sony bezogen werden (siehe [6]. Dort findet sich auch eine detaillierte FAQ-Liste zu diesem Thema.

4.2.2.1 Memorystick-Image und WLAN-Konfiguration

Der erste Schritt zur Verwendung des AIBO Remote Frameworks ist das Aufspielen eines neuen Images auf den 16 MByte großen **Programmier-Memorystick** des ERS-7 Roboters. Im Anschluss daran muss das WLAN-Modul des Roboters neu konfiguriert werden.

Zum Aufspielen des neuen **Memorystick-Images** wird der von Sony bereitgestellte **PCMCIA-Memorystick-Adapte**r (Abbildung 4.4) verwendet. Legen sie den Adapter mit dem programmierbaren Memorystick in ein Notebook ein und löschen sie die darauf befindlichen Verzeichnisse. Formatieren sie den Stick jedoch NICHT! Entpacken sie das Archiv mit dem AIBO Remote Framework Image in ein leeres Verzeichnis und kopieren sie die darin befindlichen Ordner und Dateien auf den Memorystick.



Abbildung 4.4: Sony Memorystick und PCMCIA-Adapter

Im nächsten Schritt werden die WLAN Einstellungen vorgenommen. Benutzen sie dazu die mit dem AIBO Roboter ausgelieferten Konfigurationsmanager „AIBO WLAN-Manager“ und folgen sie den Instruktionen der Online-Hilfe.

Es hat sich, während der Evaluation des AIBO Remote Frameworks in Version Beta 1, herausgestellt, dass nicht ohne Weiteres eine „Athok“-Verbindung zwischen

einer WLAN-Karte und dem Roboter aufgebaut werden konnte; der Versuch führte zum Absturz des Frameworks auf Seite des Roboters. Es ist daher zu empfehlen einen WLAN-Accesspoint zu verwendet, auch wenn die Kommunikation lediglich zwischen nur einem PC und nur einem Roboter stattfinden soll.

4.2.2.2 VAIBO-Server

Der **VAIBO-** oder **„Virtual AIBO"-Server** ist eine PC Server-Applikation, welche die Kommunikation zwischen den AIBO Robotern und den, auf dem AIBO Remote Framework basierenden, PC-Anwendungen organisiert. Die Server-Anwendung muss vor Nutzung des Frameworks installiert und gestartet werden. Abbildung 4.5 zeigt einen Screenshot des Status-Fensters des Virtual AIBO Servers. Hier werden alle aktiven Verbindungen zwischen Anwendungen und AIBO Robotern aufgelistet.

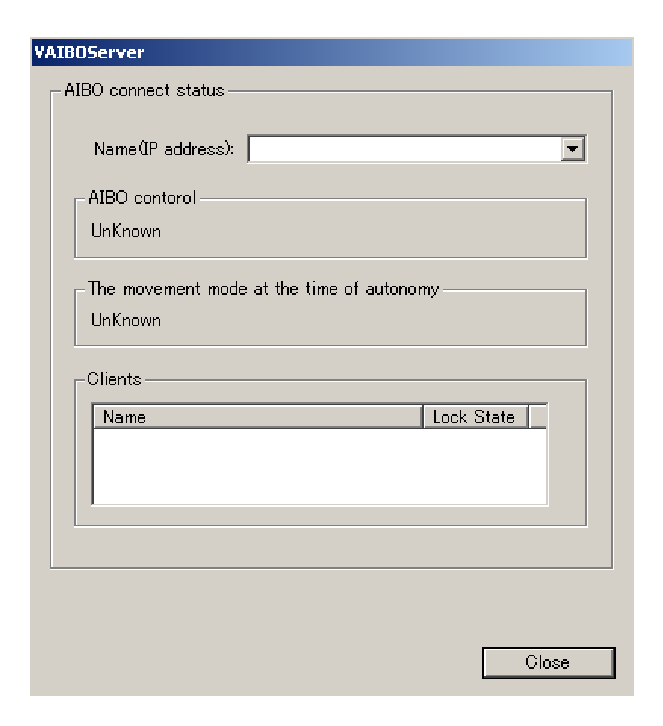


Abbildung 4.5: Virtual AIBO Server

4.2.2.3 Einbindung in MS VC++

Nachdem das AIBO Remote Framework SDK als Dateiarchiv bezogen wurde, sollte es am besten in ein neues Verzeichnis auf dem C Laufwerk des Rechners, z.b. „*c:\\AIBO_RMF*“ entpackt werden.

Das AIBO Remote Framework SDK besteht aus vier Unterordnern. In den Ordnern „*docE*“ und „*docJ*“ befindet sich eine oberflächlich gehaltene englische bzw. japanische Dokumentation. Im Grunde sind es nur Hinweise zur Architektur des Frameworks. Die anderen beiden Verzeichnisse „*INCLUDE*“ und „*LIB*“ enthalten wichtige Compiler- und Systemdateien. Im „*LIB*“-Verzeichnis befinden sich, neben den Compilerbibliotheken, die **Dynamic-Link-Libraries** (DLLs) mit den Laufzeitkomponenten des AIBO Remote Frameworks. Es empfiehlt sich, alle „**.dll*“ Dateien in das Windows Systemverzeichnis zu kopieren um allen Anwendungen Zugriff auf das Framework zu gewähren. Alternativ können die Laufzeitbibliotheken in das Programmverzeichnis der jeweiligen Anwendungen kopiert werden.

Im nächsten Schritt wird das AIBO Remote Framework SDK in die Entwicklungsumgebung des Visual C++ Compilers eingebettet. Dazu geht man wie in Abschnitt 4.2.1, im Zusammenhang mit dem Direct X SDK beschrieben, vor.

Die Liste der *INCLUDE*-Verzeichnisse des „MS VC++“ muss um das *INCLUDE*-Verzeichnis des AIBO Remote Frameworks erweitert werden, damit der Compiler die entsprechenden Header-Dateien findet. Ebenfalls muss die Liste der Linkerbibliotheken-Verzeichnisse um das *LIB*-Verzeichnis des Remote Frameworks ergänzt werden.

Sind die Verzeichnisse in die Verzeichnispfade der Entwicklungsumgebung aufgenommen, müssen lediglich noch die jeweiligen Projekteinstellungen ergänzt werden. Das heißt, die Compilerbibliotheken des AIBO Remote Frameworks:

„AIBO3D.lib CpcInfo.lib ijl15.lib VAIBOClient.lib VAIBOTTS.lib VAIBOUPnP.lib“

müssen dem Projekt hinzugefügt werden. Damit ist die Installation des AIBO Remote Framework SDKs abgeschlossen.

4.2.2.4 Beispielprogramme

Zum Test des AIBO Remote Frameworks liefert Sony einige Beispielprogramme. Mittels dieser Programme lässt sich prüfen, ob alle Einzelkomponenten des Frameworks korrekt arbeiten. Durch Kompilation des Beispielquellcodes kann außerdem festgestellt werden, ob die Entwicklungsumgebung richtig eingerichtet wurde.

Abbildung 4.6 zeigt ein Beispielprogramm, welches die Daten der AIBO Sensoren ausliest und auf dem Bildschirm eines lokalen PCs anzeigt.

Die Beispielquellcodes stellen zu einem großen Teil die Dokumentation des Remote Frameworks dar. Bedauerlicherweise verzichteten die Sony Entwickler auf aussagekräftige Kommentare und die Erläuterung wichtiger Zusammenhänge in den Quellcodes. Dadurch wird die Arbeit mit dem Remote Framework etwas erschwert.

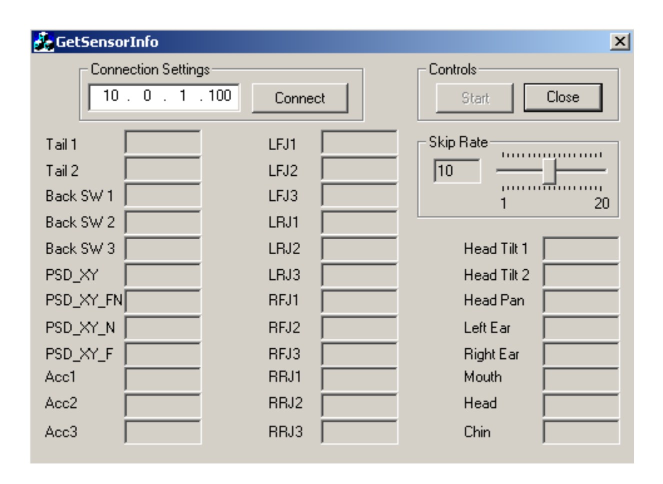


Abbildung 4.6: Beispielprogramm - Anzeige der AIBO Sensordaten

4.2.3 Track IR Enhanced SDK

Das **„Track IR Enhanced SDK“** ist ein Entwicklungspaket für den **„Track IR“-Headtracker** der Firma „Natural Point“. Der Bezug des SDKs gestaltet sich sehr bürokratisch und somit aufwendig. Das SDK muss schriftlich mittels Brief oder Fax bei der amerikanischen Firma beantragt werden. Natural Point verlangt präzise Angaben über das Projekt und das Unternehmen, in dem das

SDK eingesetzt werden soll. Nach Prüfung des schriftlichen Antrags wird per Email ein Link versendet, über den das SDK aus dem Internet heruntergeladen werden kann. Qualität und Dokumentation des Entwicklungspaketes sind jedoch vorbildlich. Der zum Bezug nötige Aufwand lohnt sich also durchaus.

4.2.3.1 Verwendung des Track IR Enhanced SDKs

Der Funktionsumfang des Track IR Enhanced SDKs ist so gering, dass Natural Point keine speziellen Compilerbibliotheken mitliefert. Stattdessen wird der direkte Zugriff auf die Laufzeitbibliotheken des Track IR Headtrackers in Form eines Beispielquellcodes beschrieben.

Die einzelnen Funktionen einer Dynamic-Link-Library, sie repräsentiert die einzige Laufzeitkomponente des SDKs, werden durch die, in einer mitgelieferten *INCLUDE*-Datei, enthaltenen Makros gekapselt. Das Ansprechen des Headtrackers in eigenen Programmen gestaltet sich, nach Studium von Dokumentation und Beispielquellcodes, einfach und effizient und bedarf keiner weiteren Erläuterung. [16]

4.2.4 NVidia Stereo Treiber

Der **„NVIDIA Stereo Treiber“** stellt eine Erweiterung der herkömmlichen NVIDIA Grafik-Chipsatztreiber, in Form eines Stereoskopie-Zusatztreibers, dar. Er kann kostenlos über die Internetseiten der Firma NVIDIA[2] bezogen werden. Es ist darauf zu achten, dass die Version des Stereo Treibers mit der Version des verwendeten Grafiktreibers übereinstimmt.

4.2.4.1 Installation

Die Installation erfolgt durch einfaches Einspielen des Treibers nach vorheriger Installation der NVIDIA Referenz Grafik-Chipsatztreiber. Das genaue Vorgehen ist auf den Internetseiten von NVIDIA dokumentiert.

[2] www.nvidia.com

4.2.4.2 Konfiguration

Der Stereo-Treiber wird über die erweiterten Grafikeinstellungen des Windows-Systems konfiguriert. In den Konfigurationsmenüs werden diverse Stereoeigenschaften eingestellt und getestet.

Wichtig für den Betrieb mit den „I-Glasses“ VR-Display sind folgende Einstellungen:

1. Der Pageflipping-Modus muss aktiviert sein.

2. Die Stereo-Bildwiederholrate sollte auf max. 100 Hz eingestellt werden. Eine höhere Rate könnte das angeschlossene Display überfordern.

3. Der Wert für die Intensität des Stereoeffektes muss auf genau 60 % eingestellt werden.

Alle anderen Werte können nach Wunsch angepasst oder in der Defaulteinstellung beibehalten werden. Abbildung 4.7 zeigt einen Screenshot des Konfigurationsdialogs des NVIDIA Stereo Treibers.

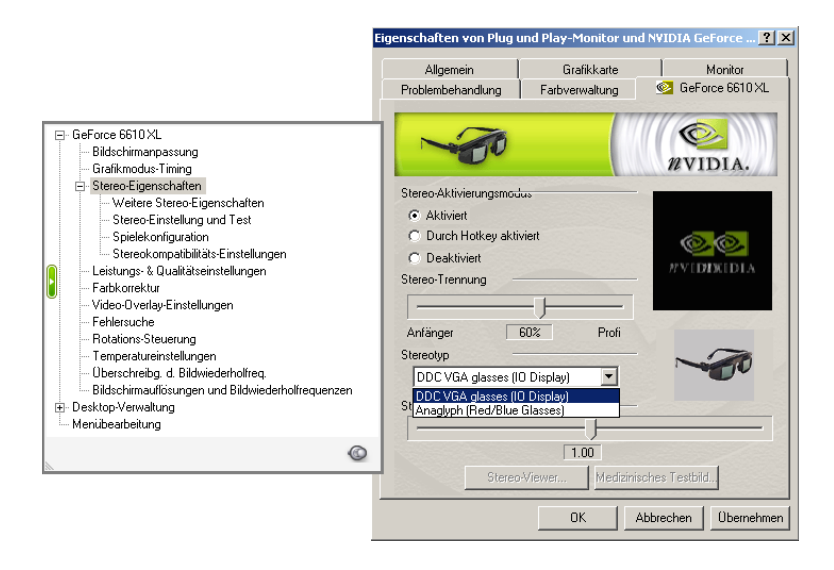


Abbildung 4.7: NVIDIA Stereo Treiber - Konfigurationsdialog

Die oben genannte Einstellung des Wertes für die Stereointensität ist zwingend, da die Software des hier entwickelten Stereokamerasystems auf exakt diesen Wert

abgeglichen wurde. Eine falsche Einstellung führt zu ungewollten Effekten bei der stereoskopischen Bildwiedergabe. [15]

4.3 AIBO Remote Framework

Das **AIBO Remote Framework SDK** ist ein Entwicklungspaket, das PC-Anwendungen ermöglicht über Wireless-LAN auf den AIBO Roboter zuzugreifen und Daten an ihn zu senden bzw. von ihm zu empfangen. Im folgenden werden wichtige Funktionen des Frameworks beschrieben und zugleich dargelegt, wie mit Hilfe dieser Funktionen ein Fernsteuersystem für den AIBO ERS-7 realisiert wird.

Die relativ detaillierten Ausführungen auf den folgenden Seiten sollen zusätzlich die dürftige Dokumentation des Frameworks ergänzen und den interessierten Leser in die Lage versetzen eigene AIBO Remote Framework-Anwendungen zu entwickeln, ohne zuvor die mitgelieferten Beispielprogramme bis ins Detail zerpflücken und analysieren zu müssen.

4.3.1 Grundlagen

Zunächst sollen einige Grundlagen über die Architektur und die Kommunikationswege des AIBO Remote Frameworks vermittelt werden.

4.3.1.1 Programmmodule

Eine AIBO Remote Framework-Anwendung setzt sich immer aus drei Modulen zusammen [12]:

1. Die **Benutzer-Anwendung**: Eine Applikation, die das Framework zur Steuerung des AIBO nutzt.

2. Das **AIBO Remote Framework API**: Bestehend aus seinen Programmbibliotheken.

3. Der **Virtual-AIBO-Server**: Eine Serveranwendung, welche die AIBO-Roboter mit dem AIBO Remote Framework verbindet.

Der Virtual-AIBO-Server wird nur einmal gestartet und kann die Verbindungen zwischen mehreren AIBO Robotern und mehreren Remote-Anwendungen verwalten. Abbildung 4.8 stellt die Kommunikation zwischen PC-Anwendungen und Robotern über das Remote Framework dar.

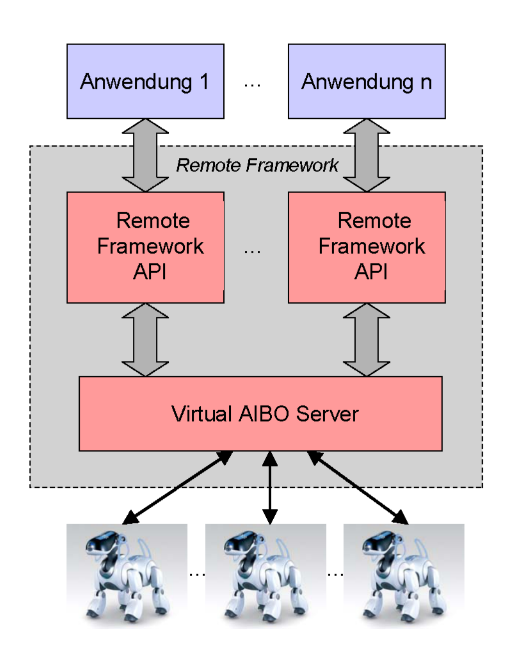


Abbildung 4.8: Kommunikation über das Remote Framework [12]

Die **Laufzeitumgebung** des Remote Frameworks besteht aus fünf dynamischen Bibliotheken (DLLs), die, als Ganzes gesehen, den Funktionsumfang des kompletten Frameworks repräsentieren. Jede dieser Bibliotheken deckt ein bestimmtes Aufgabengebiet ab:

1. **Die VAIBOClient.dll:** Enthält alle Funktionen, die zur Fernsteuerung eines AIBO ERS-7 erforderlich sind. Dazu gehören Verbindungsfunktionen, Funktionen zur Bewegungssteuerung (Laufen, Haltungsänderung, usw.) und Funktionen zur Nutzung der **Image-, Audio- und Sensorservices**, die direkt auf dem Roboter laufen, um den Datenaustausch zwischen Roboter und Remote Framework zu ermöglichen.

2. **Die AIBO3D.dll:** Ermöglicht die Anzeige eines 3D-Modells des Roboters auf dem PC-Bildschirm.

3. **Die CPCInfo.dll:** Dient zur Abfrage der Hardwareinformationen eines Roboters.

4. **Die VAIBOUPnP.dll:** Ermöglicht die Suche eines bestimmten Roboters innerhalb des Netzwerks.

5. **Die VAIBOTTS.dll:** Enthält Funktionen zum Ansprechen der **AIBO Text-To-Speach-Engine**; sie ermöglicht die Ausgabe von Sprache über den Lautsprecher des Roboters.

Die VAIBOClient.dll[3] dient als Repräsentation des entfernten Roboters auf dem lokalen PC. Sie wird dadurch zur „Kernbibliothek" des Frameworks.

4.3.1.2 Funktionen der VAIBOClient.dll

Die **VAIBOClient-DLL** stellt ein Application Interface dar, das alle wichtigen Funktionen zur Entwicklung eines Fernsteuersystems für den AIBO ERS-7 enthält. Alle in den folgenden Abschnitten beschriebenen Methoden zur Steuerung des Roboters basieren auf dieser Bibliothek.

Die VAIBOClient-DLL implementiert folgende Funktionsgruppen des Frameworks:

Verbindungskontrolle: Auf-/Abbau einer Verbindung zu einem Roboter, sowie das Sperren-/Entsperren einer Verbindung um einer Anwendung die exklusive Kontrolle über einen AIBO Roboter zu übergeben.

Kontrolle der AIBO-Services: Direkt auf dem Roboter laufen verschiedene Serverdienste zur Realisierung einer Datenkommunikation (Ton-, Bild, Sensordatenübertragung) zwischen dem Roboter und dem lokalen PC.

Abfrage der AIBO-Semantics: Semantics sind Informationen über die Änderung innerer Zustände des Roboters, beispielsweise die Änderungen des Akkuladezustands oder die Aktualisierung der Sensordaten.

[3]VAIBO = Virtual AIBO: Repräsentation eines über das Remote Framework angeschlossenen AIBO Roboters auf dem lokalen PC.

Abfrage von Sensordaten: Daten der AIBO-Sensoren können angefordert werden.

Kontrolle der Bewegungen: Die Bewegungen des Roboters (Laufen, Kopfbewegung, Haltung, usw.) können gesteuert werden. Auch vordefinierte Bewegungsabläufe können auf dem Roboter abgespielt werden.

Kontrolle über Autonomous Objects: Zu den Autonomous Object Services gehören die Spracherkennung, Verfolgung beweglicher Objekte, Gesichtssuche, Gesichtserkennung, Erkennung von Bildmustern, Zielobjekt Erkennung, Erkennung von Abgründen und das automatische Aufsuchen der Ladestation.

4.3.1.3 Kommunikation

Die Benutzer-Anwendung kommuniziert nicht direkt über das Netzwerk mit einem AIBO Roboter. Stellvertretend wird der Virtual-AIBO-Server über die Laufzeitbibliotheken des Remote Frameworks von der Anwendung angesprochen. Der Virtual-AIBO-Server organisiert die korrekte Weiterleitung von Daten und Steuerkommandos zwischen PC-Anwendungen und Robotern (vergl. Abb. 4.8).

Abbildung 4.9 zeigt die wichtigsten Teile der Architektur und Infrastruktur einer Remote Framework Anwendung. Eine vollständige, aber entsprechend komplexere schematische Abbildung kann der „AIBO Remote Framework Spezification“ [12] entnommen werden.

Die Kommunikation von der PC-Anwendung zum Roboter hin erfolgt, aus der Sicht des Anwendungsprogrammierers, durch den Aufruf von Funktionen der VAIBOClient-DLL. Die DLL-Funktionen sprechen ihrerseits den Virtual-AIBO-Server an, der die Informationen zum entsprechenden Roboter leitet.

Etwas komplizierter gestaltet sich die Kommunikation vom Roboter hin zur PC-Anwendung. Der, mit der PC-Anwendung verknüpfte, Roboter sendet seine Information an den Virtual-AIBO-Server. Der Server seinerseits nutzt das Windows-Nachrichtensystem um die Information an die entsprechende PC-Anwendung weiterzuleiten. Die Anwendung muss dazu der VAIBOClient-DLL ein Handle auf ihr Anwendungsfenster übergeben. Die vom Virtual-AIBO-Server eingehenden Nachrichten werden über die VAIBOClient-DLL an die Windows-Nachrichtenschleife

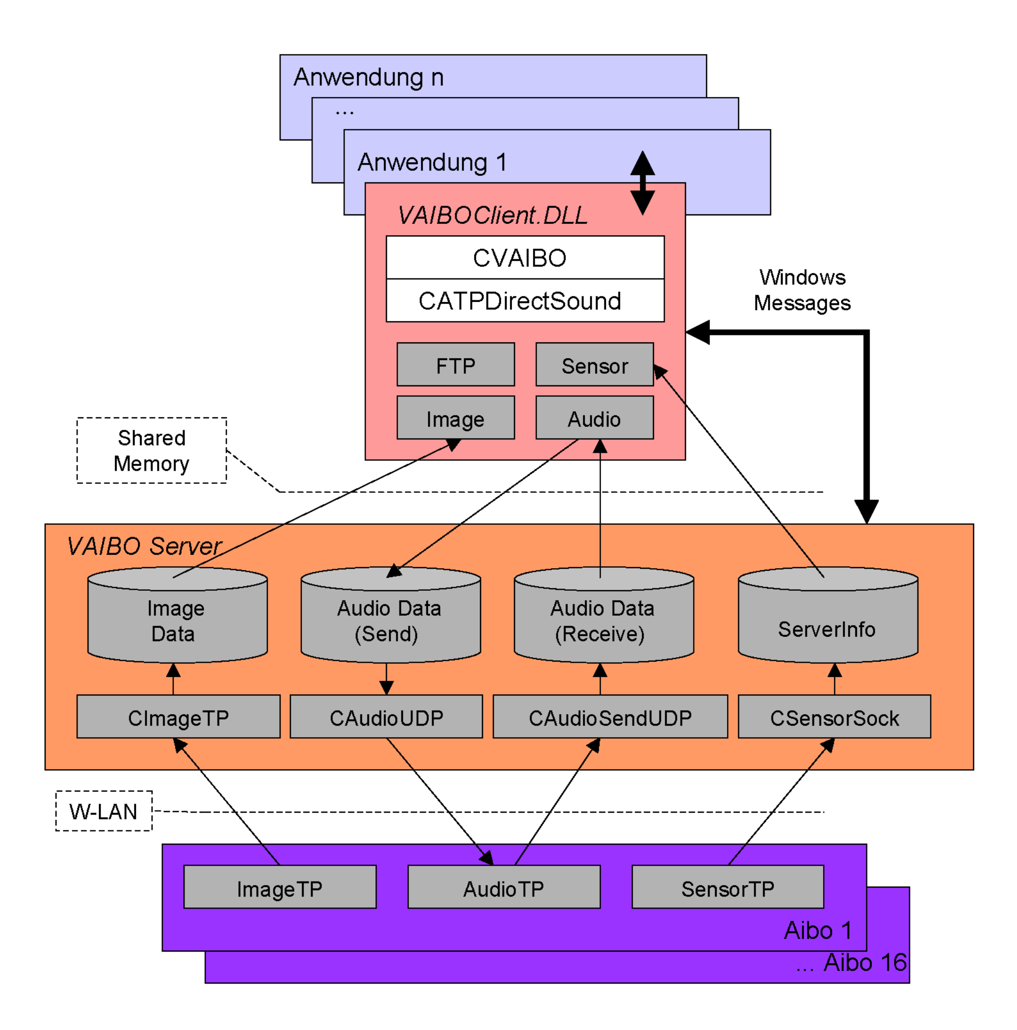


Abbildung 4.9: Architektur und Infrastruktur einer Remote Framework Anwendung [12]

der Anwendung weitergeleitet und müssen in deren Ereignisbehandlungsroutine verarbeitet werden.

Der Austausch größerer Datenmengen (Bild-, Tondaten), erfolgt nicht über die Windows-Nachrichtenschleife. Diese Daten werden über einen gemeinsamen Speicherbereich (**Shared Memory**) zwischen den Prozessen ausgetauscht. Soll beispielsweise ein Datenblock eines Audiostreams von der PC-Anwendung angenommen werden, so erhält diese die Nachricht *WM_AUDIOTP_UPDATE*. Dadurch wird signalisiert, dass ein neuer Datenblock zur Abholung aus dem Shared Memory bereit steht. Ein Zeiger auf den Datenblock wird über die, der Nachricht zugeordneten, Variable *lParam* übermittelt.

4.3.2 Die Klasse *CAiboRF*

Im Rahmen dieser Diplomarbeit wurde neben dem Stereokamerasystem eine Fernsteuerung für den AIBO ERS-7 entworfen und implementiert. Sie ermöglicht neben der Steuerung des Roboters die Abfrage wichtiger Sensordaten und die Nutzung der Mikrofone und des Lautsprechers des ERS-7 zur Übertragung von Audiosignalen.

Alle dazu nötigen Funktionalitäten wurden in der „C++“-Klasse *CAiboRF*[4] gekapselt.

Diese Klasse beinhaltet den gesamten Quellcode des im Rahmen der Arbeit entwickelten Fernsteuersystems. Das Fernsteuermodul kann daher leicht aus dem Projekt extrahiert und unabhängig vom Stereokamerasystem verwendet werden.

4.3.3 Verbindungsaufbau

Im Folgenden wird die Erstellung eines *CVAIBO* Objekts und der damit organisierte Aufbau einer Verbindung zwischen einer lokalen Remote Framework Anwendung und einem entfernten AIBO Roboter beschrieben.

4.3.3.1 *CVAIBO* Objekt

Um mit dem AIBO Remote Framework arbeiten zu können muss als Erstes eine Schnittstelle zur VAIBO-DLL generiert werden.

Die VAIBO-DLL verfügt über zwei Interfaces, welche in Form von „C++“-Klassen gekapselt werden können:

CVAIBO: Enthält alle Funktionen zur Steuerung des AIBO ERS-7 Roboters.

CATPDirectsound: Ist lediglich eine zusätzliche Schnittstelle, die das Aufnehmen und Wiedergeben von Audiosignalen über den lokalen PC vereinfacht. Dazu wird die Direct X Sound API (Directsound) verwendet.

[4] Der Quellcode der Programmklasse findet sich auf der Begleit CD zu dieser Diplomarbeit.

Die VAIBOClient-DLL wird durch die, mit dem AIBO Remote Framework mitgelieferten Compilerbibliotheken und Headerdateien, gekapselt. Der Zugriff auf die DLL gestaltet sich dadurch sehr einfach.

Die Erstellung eines Interfaces der CVAIBO-DLL geschieht schlicht durch die Konstruktion eines Objektes der Klasse *CVAIBO*. Sollen die Audio-Services des Roboters verwendet werden, wird zusätzlich ein Objekt der Klasse *CATPDirectsound* erstellt.

Im Quellcode des hier entwickelten Fernsteuersystems (Klasse *CAiboRF*) wird das oben beschriebenen Interface, *CVAIBO*, in der Methode *CAiboRF::Init(..)* erstellt und ein Zeiger auf das Interface-Objekt in der Membervariablen *m_pVAIBO* abgelegt (Abbildung 4.10).

```
bool CAiboRF::Init(HWND hWnd)
{
  if ( m_pVAibo == NULL )
  {
    m_MsgHwnd = hWnd;

    // Get CVAIBO-Interface
    m_pVAibo = new CVAIBO( m_MsgHwnd );

    return true;
  }
  // More initializations
  ...
  return false;
}
```

Abbildung 4.10: Listing-Erstellen der CVaibo-Schnittstelle

Dem Konstruktor der Klasse *CVAIBO* wird als Parameter ein Handle auf das Anwendungsfenster übergeben. Dies ermöglicht dem Schnittstellenobjekt die von den Robotern eingehenden Nachrichten in die Nachrichtenschleife der Anwendung weiterzuleiten.

4.3.3.2 Connect/Disconnect

Nachdem ein Objekt der Klasse *CVAIBO* konstruiert wurde muss über die Methode *CVAIBO::Connect(LPCTSTR IP_Address)* eine Verbindung mit einem

im Netzwerk befindlichen AIBO Roboter aufgebaut werden. Als Parameter erhält die Methode die IP-Adresse des Roboters, mit dem die Verbindung hergestellt werden soll. Als Rückgabewert liefert die Methode eine **Verbindungs-ID**. Wird ein Wert kleiner 0 zurückgeliefert, so ist der Verbindungsaufbau fehlgeschlagen. War der Verbindungsaufbau erfolgreich, erhält die Anwendung außerdem die Windows-Nachricht *WM_ VAIBO_ CONNECT*. In der zur Nachricht gehörenden Variablen *wParam* wurde, als Referenz zur Nachrichtenquelle, die entsprechende Verbindungs-ID abgelegt. Die Nachricht signalisiert, dass das CVAIBO-Objekt nun mit einem realen Roboter verknüpft ist. Das Objekt repräsentiert den entfernten Roboter innerhalb der lokalen Anwendung; es stellt also einen **„virtuellen" AIBO Roboter** dar (Virtual AIBO).

Durch Aufruf der Methode *CVAIBO::Disconnect()* eines mit einem Roboter verknüpften CVAIBO-Objekts kann eine bestehende Verbindung abgebaut werden. Bei erfolgreichem Abbau der Verbindung erhält die Anwendung die Nachricht *WM_ VAIBO_DISCONNECT*. Sollte diese Nachricht eingehen, ohne dass ein Verbindungsabbau explizit angeordnet wurde, so ist die Verbindung vermutlich unterbrochen worden (Roboter außer Reichweite des W-LAN, Störung, o.ä.).

4.3.3.3 Lock/Unlock

Der Zugriff einer lokalen Anwendung auf einen Roboter ist nicht exklusiv. Das heißt, mehrere AIBO Remote Framework-Anwendungen können Steuerkommandos an denselben Roboter senden.

Soll der Anwendung ein exklusiver Zugriff auf den Roboter gewährt werden, so muss die Verbindung durch Aufruf der Methode *CVAIBO::Lock()* gesperrt werden.

Durch Aufruf der Methode *CVAIBO::UnLock()* wird die Verbindung entsperrt.

4.3.3.4 Internal/External Control

Die AIBO Roboter verfügen unter dem Remote Framework über zwei verschiedene **Kontrollmodi**. Nach Aufbau einer Verbindung ist ein Roboter im Modus **„Internal Control"** (interne Kontrolle). In diesem Modus verarbeitet er kei-

ne Kontrollkommandos, die von Außen (external) kommen, mit Ausnahme der Einstellung der Kopfwinkel und einfachen Lauf-Befehlen („walking“).

Zum Umschalten des Betriebsmodus auf **externe Kontrolle** (external control) wird die Methode *CVAIBO::ExternalControl()* des CVAIBO-Schnittstellenobjekts aufgerufen.

Um den Roboter auf interne Kontrolle (internal Control) zu schalten dient die Methode *CVAIBO::InternalControl()*.

4.3.4 Bewegungssteuerung

Standardisierte Bewegungsabläufe des AIBO, wie das Laufen („walking“), Ausrichtung des Kopfes und Änderung der Haltung (liegend, sitzend, stehend, usw.), können direkt über die Framework-API gesteuert werden. Das „Schießen“ mit den Vorderläufen oder das „Wegstoßen“ des AIBO-Balls mit dem Roboterkopf wird ebenfalls direkt über die API gesteuert.

Weitere vordefinierte Bewegungsabläufe können durch Übermittlung einer **MW-CID** (Middle Ware Command ID) angestoßen werden. Ihr Aufbau wird in einem späteren Abschnitt besprochen.

4.3.4.1 Laufen

Eine Laufbewegung wird durch Aufruf der Methode *CVAIBO::DoWalking(int walk, int angle = 0)* angestoßen. Der erste Parameter der Methode ist die **Walking-ID**. Sie beschreibt die Art der auszuführenden Bewegung. Tabelle 4.1 zeigt einige Beispiele möglicher Walking-IDs. Der zweite Parameter ist optional und bezeichnet einen Winkel in Grad. Wird ein von 0 abweichender Winkel angegeben, bewegt sich der Roboter auf einem Kreisbogen. Durch Angabe von negativen Werten beschreitet der Roboter eine Linkskurve, positive Werte rufen dagegen das Beschreiten einer Rechtskurve hervor.

Die möglichen Werte für die Walking-ID zur Beschreibung der Bewegungsart sind in der Header-Datei *CVAIBO.h* unter *„DoWalking parameters“* definiert und können dort nachgeschlagen werden.

Tabelle 4.1: Einige Beispiele möglicher Walking-IDs [12]

Walking-ID	Bewegung des AIBO-Roboters
DW_FORWARD	Vorwärtslauf
DW_BACK	Rückwärtslauf.
DW_TURN_LEFT	Linksdrehung auf der Stelle
DW_TURN_RIGHT	Rechtsdrehung auf der Stelle
DW_STOP	Abburch einer Bewegung, Anhalten
DW_FORWARD_CNORMAL	Vorwärtslauf im „Hundgang“
DW_BACK_CNORMAL	Rückwärtslauf im „Hundgang“
DW_TURN_LEFT_CNORMAL	Linksdrehung im „Hundegang“
DW_TURN_RIGHT_CNORMAL	Rechtsdrehung im „Hundegang“

Alle Lauf- bzw. Drehbewegungen können in der Regel in den drei Geschwindigkeitsstufen „slow“, „normal“ und „fast“ ausgeführt werden. Die Angabe der Geschwindigkeitsstufe erfolgt durch einen Zusatz in der Bezeichnung der Walking-ID. Ein schneller Vorwärtslauf wird beispielsweise durch die Walking-ID *DW_FORWARD_FAST* beschrieben. Der Befehl für den Lauf einer „leichten“ Linkskurve (10 Grad) im schnellen Gang lautet demnach beispielhaft *DoWalking(DW_FORWARD_FAST, -10).*

4.3.4.2 Kopfwinkeleinstellung

Die Einstellung des Blickwinkels des AIBO-Kopfes erfolgt mittels der Methode *CVAIBO::ChangeHeadAngle(int angleH, int angleV, int nSpeed = 2)* des CVAIBO-Objekts. Der erste Parameter, *angleH,* beschreibt den Blickwinkel in horizontaler Ebene, der zweite Parameter, *angleV*, den Blickwinkel in vertikaler Ebene. Die Winkelangaben erfolgen jeweils in Grad innerhalb eines Intervalls von [-93°, ..., 0°, ..., +93°] für den Horizontalwinkel bzw. [-20°, ..., +50°] für den Vertikalwinkel. Wird für beide Winkel ein Wert von 0° angegeben, blickt der Roboter nach vorne in Richtung seiner Längsachse. Bei Angabe negativer Winkel dreht er den Kopf nach links bzw. nach oben. Bei Angabe positiver Werte dreht AIBO den Kopf nach rechts bzw. unten.

Der dritte Parameter, *nSpeed,* ist optional. Er gibt die Geschwindigkeit der Bewegung an. Leider ist dieser Parameter nicht näher dokumentiert, so kann die gewünschte Einstellung nur experimentell herausgefunden werden.

4.3.4.3 Motions

Der AIBO ERS-7 ist in der Lage vordefinierte, auf dem Memorystick des Roboters abgelegte Bewegungsabläufe, sogenannte **Motions**, wiederzugeben. Ein solcher Bewegungsablauf könnte beispielsweise ein Tanz oder eine automatische 180°-Wende sein. Erstellt werden können Motions unter zu Hilfenahme des AIBO Motion Editors.

Angestoßen wird die Wiedergabe vordefinierter Motions durch Aufruf der Methode *CVAIBO::PlayMotion(LPARAM lmsCmd).* Der angegebene Parameter ist das sogenannte **L-M-S-Command**. Die Abkürzung L-M-S steht für Large/-Middle/Small und bezeichnet lediglich die Aufteilung des 32-Bit breiten Kommandowertes in einzelne Bytes (Large-/Middle-/Small-Byte). Tabelle 4.2 zeigt die Unterteilung des Kommandowertes in L-ID, M-ID, S-ID und Level-Wert. Der **Level-**Wert beschreibt eine Art Kontext unter dessen Voraussetzung der Bewegungsablauf angestoßen werden kann. So können beispielsweise in Abhängigkeit davon, ob der Roboter sich in einer stehenden oder liegenden Position befindet, unterschiedliche Bewegungsabläufe angestoßen werden.

Tabelle 4.2: LMS-Command [12]

4. Byte	3. Byte	2. Byte	1. Byte
L-ID	M-ID	S-ID	Level

Die L-, M- und S-ID strukturieren die wiederzugebenden Bewegungen in eine dreistufige Hierarchie. Zur Erklärung der Methode soll ein Beispiel herangezogen werden. Wir nehmen an die IDs würden auf folgende vordefinierte Makros gesetzt:

- L-ID = E_MOVEPOS
- M-ID = M_LIE

- S-ID = S_SLEEPM

Die **Makros** werden in der Exel-Datei *RFW_CDB_beta1.xls* des AIBO Remote Frameworks näher beschrieben.

Die L-ID gibt die Bewegungsgruppe *E_MOVEPOS* an. Das Makro steht für „Move Posture", was soviel bedeutet wie „Ändere Haltung". Die M-ID legt die Untergruppe *M_LIE* fest. Es soll also eine liegende Position eingenommen werden. Am Ende gibt die S_ID an, wie diese Position genau einzunehmen ist. Das Makro *S_SLEEPM* steht für „Bone Pillow Sleeping Posture", also „Knochenkissen-Schlafposition". Der AIBO Roboter wird demnach, falls sein AIBO-Knochen vor ihm liegt, sich hinlegen und seinen Knochen als Schlafkissen benutzen.

4.3.5 Sensorabfrage

Die Abfrage der internen Sensoren (Entfernungssensoren, Batteriesensoren, Tastsensoren, usw.) des AIBO Roboters erfolgt über einen auf dem Roboter laufenden Servicetask. Alle Sensordaten werden, als Datenpaket, in vorher definierten Zeitintervallen der Anwendung zugestellt bzw. können von dieser explizit angefordert werden.

4.3.5.1 AIBO Sensor TP

Die **Sensor-TP** ist der Service der, innerhalb des AIBO Remote Frameworks, für die Abfrage der Sensoren und Zustellung der Sensordaten über das Netzwerk verantwortlich ist.

Zur Aktivierung der Sensor-TP und dem Empfang von Sensordaten sind folgende Schritte nötig:

1. Anforderung des entsprechenden Netservices zwecks Aktivierung der Sensor-TP.

2. Setzen der „Skip-Rate", der Frequenz, mit der die Datenpakete der Anwendung zugestellt werden.

3. Start des Datentransfers.

Im ersten Schritt muss der, zur Sensor-TP gehörige, **Net-Service** gestartet werden. Dieser Net-Service läuft als Task auf dem Roboter und ist für das Abfragen der Sensoren und das Versenden der Daten verantwortlich. Zum Starten des Services bedient man sich der Methode *CVAIBO::RequestNetService(SENSOR_ID)*. Das Makro *SENSOR_ID* ist in der INCLUDE-Datei *VAIBODef.h* definiert und identifiziert den **Sensor-Net-Service**. Weitere Net-Services sind beispielsweise „FTP“-, „Audio“- und „Image“-Service.

Im zweiten Schritt wird durch Aufruf der Methode *CVAIBO::SensorSetSkipRate(int SkipRate)* die **Sensor Skip-Rate** gesetzt. Das ist die Frequenz, mit welcher die Sensordatenpakete der lokalen Anwendung zugestellt werden. Die Einheit ist Zustellungen/Sekunde.

Im dritten Schritt muss der Datentransfer gestartet werden. Dies erfolgt durch Aufruf der Methode *CVAIBO::SensorTransferStart()*. Mittels der Methode *SensorTransferStop()* wird die Datenübermittlung gestoppt.

Alle oben beschriebenen Schritte zum Starten und Schließen der Sensorservices sind im Quellcode der AIBO-Fernsteuerung in der Methode *CAiboRF::StartSensorService(bool DoStart)* zusammengefasst.

4.3.5.2 Sensordatenzustellung

Die Daten aller Sensoren werden der Anwendung im Takt der eingestellten Skip-Rate als Paket (**Sensordaten-Array**) zugestellt.

Steht ein neues Sensor-Datenpaket zur Abholung bereit, wird die lokale Anwendung durch Eingang der Windows-Nachricht *WM_VAIBO_SENSOR_DATA* benachrichtigt. Die, zur Nachricht gehörigen, Variablen *wParam* und *lParam* enthalten wichtige Informationen über das Datenpaket.

Die Variable *lParam* enthält einen Zeiger auf einen Array, welcher eine Liste von Datenstrukturen des Typs *SensorRec* (Abbildung 4.11) enthält. Jede der Datenstrukturen enthält die ID und die Daten eines bestimmten Sensors. Im *„LOWORD“* der Variablen *wParam* („*LOWORD(wParam)*“) ist die Anzahl der im Array gespeicherten *SensorRec*-Strukturen verzeichnet. In „*HIWORD(wParam)*“ steht die zur Nachricht gehörige Verbindungs-ID. Sie identifiziert den Roboter bzw. die Verbindung, von dem/der die Nachricht eingegangen ist.

```
// Sensor Data
typedef struct SensorRec {
    int             sensorID;
    unsigned long   value;
    long            percentage_x;
    long            percentage_y;
} SensorRec, *SensorRecP;
```

Abbildung 4.11: Listing-Definition der Struktur *„SensorRec“*

Ein Sensor des AIBO ERS-7 wird durch seine **Sensor-ID** identifiziert. Sollen die Daten eines bestimmten Sensors in der lokalen Anwendung ausgewertet werden, muss der *SensorRec*-Array mittels einer Schleife durchlaufen und nach der ID des gewünschten Sensors durchsucht werden. Abbildung 4.12 zeigt beispielhaft das Auslesen und das Konvertieren der Daten eines Entfernungssensors mit der ID *„PSDWithXY_FN“*[5].

```
int             numOfData, aiboID;
aiboID       = HIWORD( wParam );
numOfData    = LOWORD( wParam );

// Loop for each sensor-rec
for (int i = 0; i < numOfData; i++) {

  // Sensor value
  long lValue = (long)(((SensorRec *)recArray)[i].value);

  // receive Distancesensor data
  if(((SensorRec *)recArray)[i].sensorID == PSDWithXY_FN )
  {
     // convert and store distance data
     m_fDistanceSensor = (float)(lValue/10000.0f);
  }
}
```

Abbildung 4.12: Listing-Auswertung von Sensor-Daten

[5]Die IDs aller internen Sensoren sind in der INCLUDE-Datei *CPCInfo.h* des Frameworks zu finden.

```
// Create new DirectSound wrapper and init
m_pATPDirectSound = new CATPDirectSound();

// Initialize CATPDirectSound-Object
m_pATPDirectSound->Init( m_MsgHwnd );
```

Abbildung 4.13: Listing-Initialisierung der *CATPDirectSound*-Schnittstelle

4.3.6 Audio

Es ist über das AIBO Remote Framework möglich sowohl Audiodaten von den Mikrofonen eines AIBO Roboters zu empfangen, als auch Audiodaten über den eingebauten Lautsprecher des Roboters wiederzugeben. Die Daten werden dazu über die Netzwerkverbindung „gestreamt“.

4.3.6.1 AIBO Audio-TP

Die **Audio-TP** ist, ähnlich wie die Sensor-TP, ein auf dem Roboter laufender Service-Task, der das Streamen von Audio Daten über das Netzwerk ermöglicht.

Zur Nutzung der Audio-TP wird zunächst eine Hilfsschnittstelle, das ***CATPDirectsound*-Interface**, konstruiert. Die Schnittstelle ermöglicht das Aufnehmen und Wiedergeben von Audiodaten über das Mikrofon und die Lautsprecher eines lokalen PCs. Um die Schnittstelle zu nutzen, wird ein Objekt der Klasse *CATPDirectsound* konstruiert und Initialisiert (siehe Abbildung 4.13).

Die Audio-TP ist in zwei unterschiedliche Services aufgeteilt. Ein Service dient dem Empfang von Audiodaten von den AIBO-Mikrofonen. Der zweite ist für das Senden von Daten an den Lautsprecher des AIBO-Roboters zuständig. Die Audiodaten werden in Form eines „Stroms“ (engl. Stream) kleiner Datenblöcke übertragen.

4.3.6.2 Empfangen von Audiodaten

Die Einstellung und Aktivierung der Audio-TP zum Empfangen von Audio-Daten von den internen Mikrofonen des AIBO gliedert sich in folgende Schritte:

1. Festlegung des Audio-Formats.

2. Aktivieren der AUDIO-TP und Anforderung des Net-Services („*AUDIO_IN_ID*“).

3. Einstellung von Modus und Audio-Format.

4. Aktivierung der Directsound-Schnittstelle zur Wiedergabe des eingehenden Audiostreams.

5. Starten des Services zum Empfang von Audiodaten.

Im ersten Schritt sollte das gewünschte **Audio-Format** gewählt und in einer Variablen abgelegt werden. Das Format muss später sowohl für die Audio-TP, als auch für die Directsound-Hilfsschnittstelle eingestellt werden.

Der AIBO ERS-7 verfügt über zwei Mikrofone, die auf linker und rechter Seite des Kopfes montiert sind. Die Audio-TP bietet zur Abfrage der Mikrofone die in Tabelle 4.3 dargestellten Audio-Formate. Möglich ist jeweils eine Abtastrate von 16 bzw. 8 Kilo-Samples/sec mit einer Auflösung von 16 bzw. 8 Bit in Stereo oder Mono. Die zugehörigen Makros sind in der INCLUDE-Datei *VAIBODef.h* definiert.

Im zweiten Schritt wird die Audio-TP aktiviert und der, zum Empfangen von Audiosignalen erforderliche, Net-Service angefordert. Das Aktivieren der Audio-TP geschieht über die Methode *CVAIBO::AudioGetOpen(int mode)*. Als Parameter wird das zuvor gewählte Audio-Format übergeben. Bei Erfolg liefert die Methode den Wert des Define-Makros *AUDIOTP_NOERROR* zurück. Makros, die eventuelle Fehlermeldungen beschreiben, sind in der INCLUDE-Datei *VAIBODef.h* definiert.

Als nächstes wird der ***„AUDIO_IN“***-Net-Service angefordert. Dazu dient folgender Methodenaufruf: *CVAIBO::RequestNetService(AUDIO_IN_ID)*. Das übergebene Makro *AUDIO_IN_ID* identifiziert den Net-Service für den Audio-Empfang vom AIBO Roboter.

In Schritt drei wird der Audio-Mode und das Audio-Format eingestellt. Dies geschieht durch folgenden Methodenaufruf: *CVAIBO::SendCmd(APPCMD_AUDIO_RECV_MODE, true, m_nAudioTPMode)*. Der erste Parameter ist wieder ein Define-Makro und beschreibt ein Kommando, welches der Audio-TP mitteilt, dass der Service aktiviert werden soll, der das „Streaming“ von Audio-Daten vom AIBO zum lokalen PC ermöglicht. Der zweite Parameter, *true,* besagt,

Tabelle 4.3: Wave-Formate der Audio-TP [12]

Makro	Wave-Format
ATP16K_16BIT_STEREO	16k, 16Bit, Stereo
ATP16K_16BIT_MONO_L	16k, 16Bit, Mono links
ATP16K_16BIT_MONO_R	16k, 16Bit, Mono rechts
ATP16K_16BIT_MONO_MIX	16k, 16Bit, Mono beide
ATP8K_16BIT_STEREO	8K, 16Bit, Stereo
ATP8K_16BIT_MONO_L	8K, 16Bit, Mono links
ATP8K_16BIT_MONO_R	8K, 16Bit, Mono rechts
ATP8K_16BIT_MONO_MIX	8K, 16Bit, Mono beide
ATP16K_8BIT_STEREO	16K, 8Bit, Stereo
ATP16K_8BIT_MONO_L	16K, 8Bit, Mono links
ATP16K_8BIT_MONO_R	16K, 8Bit, Mono rechts
ATP16K_8BIT_MONO_MIX	16K, 8Bit, Mono beide
ATP8K_8BIT_STEREO	8K, 8Bit, Stereo
ATP8K_8BIT_MONO_L	8K, 8Bit, Mono links
ATP8K_8BIT_MONO_R	8K, 8Bit, Mono rechts
ATP8K_8BIT_MONO_MIX	8K, 8Bit, Mono beide

dass die Methode erst zurückkehren soll, wenn das übergebene Kommando ausgeführt wurde. Der dritte Parameter ist das gewählte Audio-Format.

Um den vom AIBO eingehenden Audiostream hörbar zu machen wird im vierten Schritt die Audiowiedergabe über die PC-Lautsprecher aktiviert. Dazu wird eine Methode der Directsound-Hilfsschnittstelle aufgerufen: *CATPDirectsound::StartPlay(int format)*. Der übergebene Parameter ist wieder das festgelegte Audio-Format.

Der Directsound-Hilfsschnittstelle mitzuteilen, das sie mit dem Abspielen des Audiostreams beginnen soll reicht selbstverständlich noch nicht aus, denn die Daten des eingehenden Streams müssen blockweise zur Audio-Ausgabe übertragen werden. Zu diesem Zweck erhält die Anwendung, jedes Mal, wenn ein neuer Datenblock zur Abholung bereit steht, die Windows-Nachricht *WM_AUDIO_UPDATE*. Die zur Nachricht gehörige Variable *lParam* enthält diesmal keinen Zeiger auf den Datenblock, sondern beinhaltet lediglich die Größe des Blocks in Bytes. Zur Abholung der Daten muss die Methode *CVAIBO::AudioGetData(unsigned char *buff, int size)* aufgerufen werden. Als Para-

meter werden ein Zeiger auf einen leeren Datenpuffer und die Größe des Puffers übergeben. Die Wiedergabe des Audio-Datenblocks auf den Lautsprechern des PC erfolgt durch Aufruf der Methode *CATPDirectsound::PlayWaveData(char* buf, int size)*. Der erste Parameter ist ein Zeiger auf den wiederzugebenden Datenblock, während der zweite Parameter die Größe des Blocks angibt.

Um die von den Mikrofonen des AIBO aufgenommenen Audio-Signale auf den PC Lautsprechern wiederzugeben muss im fünften und letzten Schritt die Übertragung des Audiostreams aktiviert werden. Dies geschieht durch übersenden eines Start-Kommandos an die Audio-TP: *CVAIBO::SendCmd(APPCMD_AUDIO_RECV_START)*.

4.3.6.3 Senden von Audiodaten

Die Einstellung und Aktivierung der Audio-TP zum Senden von Audio-Daten zur Ausgabe über den internen Lautsprechers des AIBO gliedert sich in folgende Schritte:

1. Festlegung des Audio-Formats.
2. Aktivierung der Audio-TP zum Senden von Audio-Daten und Anforderung des Net-Service („*NETAUDIO_ID*“).
3. Einstellung des Audio-Formats und Starten der Übertragung.
4. Starten der Klangaufnahme über die Soundkarte des lokalen PC (Mikrofon).

Im ersten Schritt wird, ähnlich wie beim Empfangen von Audio-Daten, das Audio-Format festgelegt. Da der AIBO Roboter nur über einen Lautsprecher verfügt und somit nicht in der Lage ist, Stereosignale wiederzugeben beschränkt sich die Bandbreite möglicher Formate auf die in Tabelle 4.3 mit dem Zusatz „*..._MONO_MIX*“ versehenen Wave-Formate.

Im zweiten Schritt wird die Audio-TP für die Übertragung von Audio-Informationen vom lokalen PC zum AIBO hin aktiviert und der zugehörige Net-Service angefordert. Dies geschieht durch Aufruf der Methode *CVAIBO::AudioSendOpen(int mode)*. Der zu übergebende Parameter ist das zuvor

festgelegte Audio-Format. Die Anforderung des Net-Services geschieht durch den Aufruf *CVAIBO::RequestNetService(NETAUDIO_ID)*. Das Define-Makro ***NETAUDIO_ID*** identifiziert den Service zur Übertragung von Audio-Daten zum AIBO hin.

In Schritt drei wird das Audio-Format zur Wiedergabe der Daten eingestellt und die Audio-Übertragung gestartet. Das Einstellen des Formats geschieht mittels der Methode *CVAIBO::AudioSendSetMode(int format)*. Der Parameter *format* ist das festgelegte Audio-Format. Um die Audio-Übertragung zu starten wird die Methode *CVAIBO::AudioSendPlay(BOOL isKutipaku = true)* aufgerufen. Der Parameter *isKutipaku* ist optional und standardmäßig auf „*true*" gesetzt. Wird für diesen Parameter „*true*" übergeben, so bewegt sich der Mund des Roboterhundes im Takt des über den AIBO-Lautsprecher ausgegebenen Signals. Werden Sprachsignale übertragen, imitiert er also die Mundbewegung des Sprechers.

Im vierten und letzten Schritt muss dafür gesorgt werden, dass an der Soundkarte des PCs anliegende analoge Audio-Signale (Mikrofon) digitalisiert und blockweise zum AIBO übertragen werden. Zum digitalisieren der Audio-Signale wird die Schnittstelle *CATPDirectsound* zu Hilfe genommen. Durch Aufruf der Methode *CATPDirectsound::StartCapture(...)* wird die Digitalisierung der Audio-Signale gestartet (siehe auch Abbildung4.14). Zur Übernahme, der von der Directsound-Hilfsschnittstelle eingehenden Audio-Blöcke, wird ein Callback-Verfahren eingesetzt. Der Methode *CATPDirectsound::StartCapture(...)* wird, neben dem festgelegten Audio-Format, ein Funktionszeiger auf eine Callback-Prozedur übergeben. Diese Prozedur muss vom Anwendungsentwickler nach einer vorgegebenen Funktionsdeklaration implementiert werden und wird von der Directsound-Hilfsschnittstelle immer dann aufgerufen, wenn ein neuer Datenblock digitalisiert wurde. Als Parameter werden der Prozedur (unter anderem) ein Zeiger auf den neuen Audio-Datenblock und die Größe des Datenblocks übergeben. Innerhalb der Callback-Prozedur wird der neue Datenblock mittels der Methode *CVAIBO::AudioSendData(char *data, int length)* über das AIBO Remote Framework zum Roboter übertragen und dort auf dem Lautsprecher wiedergegeben. Abbildung 4.14 zeigt eine exemplarische Implementierung der Callback-Capture-Prozedur.

```
...
// Start capturing PC-Audio
m_pATPDirectSound->StartCapture(m_nNetAudioMode,
GetCaptureProc, NULL);
...
// A CallBack-Function to capture PC-Audio
BOOL GetCaptureProc( void* pCaptureData,
   DWORD dwCaptureLength, BOOL isEnd, void* pParam )
{
  CVAIBO* pVaibo = CAiboRF::m_pVAibo;

  // Transmission
  int err =
    pVaibo->AudioSendData( (char*)pCaptureData,
      dwCaptureLength )

  if( err != AUDIOTP_NOERROR ) {
    // An error occured !
    return false;
  }
  return true;
}
```

Abbildung 4.14: Listing-CATPDirectSound-Capture-Prozedur

4.4 Implementierung des Stereokamerasystems

Das folgende Unterkapitel beschreibt die hard- und softwaretechnische Realisierung des Stereokamerasystems.

Zunächst werden einige wichtige Grundlagen über die verwendete Multimediaschnittstelle Direct X vermittelt und die für die Konstruktion des Kamerasystems verwendete Hardware vorgestellt. Darauf folgend wird ein Überblick über die Programmmodule der im Rahmen der Arbeit entwickelten Anwendung „Stereoview“ dargelegt und die eingesetzten Programmiertechniken detailliert beschrieben und erklärt.

4.4.1 Direct X Grundlagen

Direct X ist eine, von der Firma Microsoft entwickelte, Multimediaschnittstelle. Sie soll dem Anwendungsentwickler einen einheitlichen Zugriff auf die, in einem Rechner installierte, Multimediahardware bieten. Direct X stellt eine zusätzliche Softwareschicht zwischen einer Anwendung und den Gerätetreibern der Hardware dar. Zur Umsetzung der, im Rahmen dieser Diplomarbeit gestellten Aufgabe, der Entwicklung eines Stereokamera-Systems, wurde diese Schnittstelle in der Version

9.0 zur Aufnahme der Bilddaten und der Reproduktion dieser Daten auf einem stereoskopischen Display eingesetzt. An dieser Stelle soll dem Leser ein kurzer Überblick über die Architektur und die Verwendeten Komponenten der Direct X 9-Schnittstelle gegeben werden.

4.4.1.1 Architektur

Direct X baut auf dem **„Component Object Model“** auf, welches der Organisation der einzelnen Komponenten dient.

Das **COM** fordert die Aufteilung einer großen Programmbibliothek, wie Direct X es ist, in verschiedene Schnittstellen (Interfaces). Jede dieser Schnittstellen deckt einen mehr oder weniger großen Aufgabenbereich ab. Formal sind diese Schnittstellen nichts anderes, als „C++“-Klassen, die aber lediglich Methoden (keine Variablen) beinhalten.

Die Namen der COM-Schnittstellen beginnen mit dem Präfix „*I*“ für „Interface“. Danach folgt der eigentliche Name der Schnittstelle, z.b. bezeichnet *ID3DDevice9* eine Schnittstelle zur Steuerung eines 3D-Grafikadapters. Zu jedem Schnittstellennamen gibt es eine Zeigervariante. Diese beginnt mit dem Präfix „*P*“ (Pointer) bzw. „*LP*“ (Long Pointer), z.b. *PDIRECT3D9*.

Alle auf COM basierende Schnittstellen sind von der Basisschnittstelle ***IUnknown*** abgeleitet. Die *IUnknown*-Schnittstelle definiert alle für die Schnittstellenverwaltung erforderlichen Methoden. Mittels der Methode *IUnknown::QueryInterface(...)* der Basisschnittstelle kann ein Zeiger auf eine „höhere“ Direct X-Schnittstelle angefordert werden.

Jede COM-Schnittstelle und somit auch jede Direct X-Schnittstelle besitzt, zum Zwecke ihrer exakten Beschreibung, eine einzigartige Schnittstellen-GUID[6]. Zur Anforderung eines Zeigers auf eine Direct X-Schnittstelle wird der Methode *IUnknown::QueryInterface(...)* die GUID der gewünschten Schnittstelle übergeben. Um einen einfachen Umgang mit den GUIDs zu gewährleisten wurden deren Werte in Define-Makros abgelegt. Das Define-Makro *IID_IDirect3D9*, steht beispielsweise für die GUID der Schnittstelle *IDirect3D9*. Die Namen der Define-Makros für Schnittstellen-IDs besitzen das Präfix „*IID_*“. [2]

[6]GUID = Global Unified Identifier: Weltweit eindeutige Identifizierungsnummer

Detailliertere Informationen zur Architektur und den einzelnen Komponenten-Schnittstellen von Direct X finden sich in der „Direct X SDK Dokumentation“ [13] auf die ich an dieser Stelle verweisen möchte.

4.4.1.2 Directshow

Die Direct X Unterkomponente **Directshow** dient der Aufnahme, Verarbeitung und Wiedergabe von Multimediainhalten.

In dem im Rahmen dieser Diplomarbeit umgesetzten Projekt, dem „3D Stereokamera System“, wird die Directshow-Schnittstelle zur Konfiguration, Ansteuerung und Abfrage der Video-Eingabegeräte verwendet. Die Verarbeitung der Videodaten erfolgt nur teilweise durch Directshow-Komponenten. In großen Teilen erfolgt diese durch speziell für das Projekt entwickelte Programm-Modulen, die jedoch in die Directshow-Architektur eingebettet werden.

Die Verwendung von Directshow zur Nutzung von Videoeingabegeräten führt zu einer sehr großen Hardware-Unabhängigkeit. Das fertige Programm **„Stereo View“** ist praktisch zu allen Videoeingabegeräten kompatibel, für welche zum Betriebssystem Windows kompatible Treiber existieren.

4.4.1.3 Direct-3D

Direct-3D ist eine Direct X-Komponente, die der Darstellung dreidimensionaler Computergrafiken auf 3D-fähigen Grafikadaptern dient. Zur Entwicklung des hier vorgestellten Stereokamera-Systems wurde die Direct-3D-Schnittstelle zur Darstellung entsprechender Videodaten auf einem stereoskopiefähigen Display verwendet.

Es soll an dieser Stelle darauf hingewiesen werden, dass in Direct-3D Version 9 grundsätzlich die Darstellung stereoskopischer Bilddaten nicht vorgesehen ist. Es bleibt zu hoffen, dass ein solches Feature in einer zukünftigen Version integriert wird. Einige Hersteller von Grafikchipsätzen (ATI, NVIDIA) bieten jedoch Zusatzmodule für ihre Chipsatztreiber an, welche die stereoskopische Darstellung der unter Direct-3D gezeichneten 3D-Vektorgrafiken „erzwingen“.

Durch die Modifikation der Grafiktreiber wird die Darstellung stereoskopischer Bild- oder Videodaten über die Direct-3D-Schnittstelle, zumindest im Rahmen gewisser Grenzen, ermöglicht.

4.4.2 Eingesetzte Hardware

Zur Entwicklung eines Stereokamerasystems für den AIBO ERS-7 Roboter musste eine spezielle Stereokamera konstruiert werden. Die Reproduktion der stereoskopischen Bilddaten erfolgt auf einem Virtual-Reality-Helm des Typs „I-Glasses“ der Firma „IO-Display Systems“.

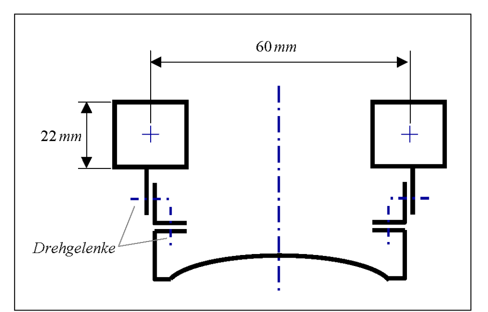
4.4.2.1 Doppelkamerasystem

Aufgrund einer relativ geringen Größe und Tragkraft des Roboters AIBO ERS-7 wurde ein **Doppelkamerasystem** aus zwei **ZT-810 bzw. ZT-811 Mini-Funk-Kameras**[7] aufgebaut.

Für die Montage der Kameras auf dem Kopf des Roboters wurde eine spezielle Halterung (Abbildungen 4.15 und 4.16) entwickelt. Sie besteht aus einem, auf die Form des Kopfes zugebogenen, Lochblechstreifen. Auf die Grundhalterung wurden zwei Winkel aufgeschraubt, an denen die Kameras montiert werden. Die Konstruktion ist so geschaffen, dass die beiden Kameras einen Horizontalabstand von etwa 6 cm besitzen (Augenabstand) und sowohl in horizontaler, als auch in vertikaler Ebene exakt ausgerichtet werden können.

Die Mini-Funk-Kameras senden die aufgenommenen Bilder im analogen PAL-Format zu zwei stationären Empfangsteilen. Die Übertragung erfolgt mit einer Sendeleistung von 10 mW auf dem, in Europa freigegebenen, 2,4 GHz-Frequenzband. Die Sender erzielen eine Reichweite, die, trotz geringerer Sendeleistung, etwa der des Wireless-LAN entspricht. Abbildung 4.17 zeigt eine Großaufnahme einer ZT-811 Kamera und den zugehörigen Receiver mit dem Wahlschalter für den Funkkanal, sowie Audio- und Videoausgang. Das aus dem Kamera herausgeführte Kabel dient lediglich der Betriebsspannungsversorgung. Das Videosignal

[7] Die Kameras ZT-810 und ZT-811 unterscheiden sich in Größe und Bildqualität. Die ZT-810 hat die dopellte Länge einer ZT-811, etwa 45 mm. Sie liefert eine etwas bessere Bild- und Übertragungsqualität. Das Doppelkamerasystem kann mit beiden Kameratypen problemlos aufgebaut werden.



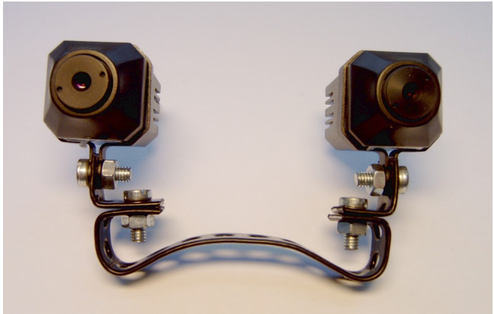


Abbildung 4.15: Kamerahalterung: Skizze



Abbildung 4.16: Kamerahalterung mit 2 ZT-810 Kameras

wird über die, im Bild deutlich erkennbare, kleine Antenne übertragen. Im vorderen, linken Teil des Bildes ist das winzige, etwa 1,5 mm im Durchmesser große, Kameraobjektiv zu sehen.

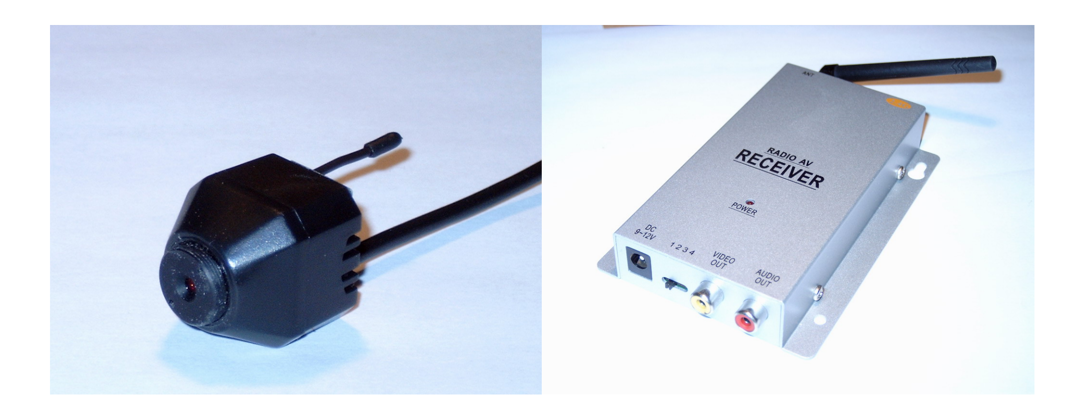


Abbildung 4.17: ZT-811 Funkkamera und Empfänger

Die von den Funkempfängern erfassten Videosignale werden durch zwei **Videocapturegeräte** des Typs **„Osprey-50 USB“** der Firma „ViewCast“ digitalisiert und via USB-Kabel an einen PC übertragen. Abbildung 4.18 zeigt eines der beiden Videocapturegeräte.



Abbildung 4.18: Osprey-50-USB Videocapturegerät

Das im Rahmen der Arbeit entwickelte Programm **„Stereo View“** ist grundsätzlich in der Lage mit allen, unter Windows betriebenen, Videocapturegeräten umzugehen. Es gilt jedoch an dieser Stelle zu beachten, dass nicht alle Videocap-

turegeräte kaskadiert werden können. In der Regel lassen die Gerätetreiber nur den Anschluss eines einzigen Gerätes des gleichen Typs zu.

Zur **Energieversorgung** der Kameras wurde ein spezieller Akkupack (Abbildungen 4.19 und 4.20) konstruiert. Die Form des aus sieben Nickelmetallhydridzellen des Typs „Micro“ bestehenden Pakets wurde zum Betrieb auf dem AIBO ERS-7 ausgelegt. Der Akku liefert eine Spannung von 8,4 Volt und besitzt eine, in Hinsicht auf Größe und Gewicht, beachtliche Kapazität von 800 mAh. Das Kamerasystem, mit einer Stromaufnahme von etwa 120 mA, wird so über einen Zeitraum von ca. 6,5 h mit Energie versorgt. Der Akkupack wird, über den zugehörigen Ladeadapter, am Netzteil eines Funkkamera-Empfängers geladen. Der Ladevorgang nimmt etwa 4 h in Anspruch.

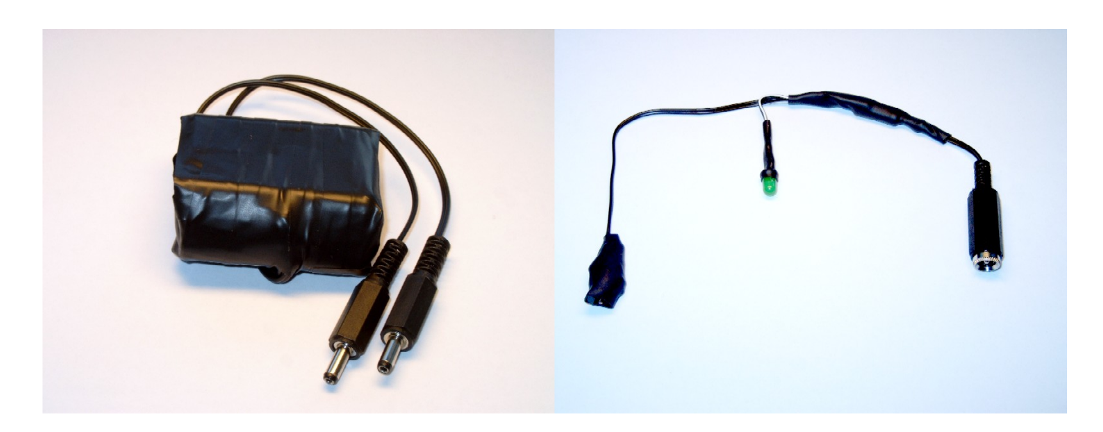
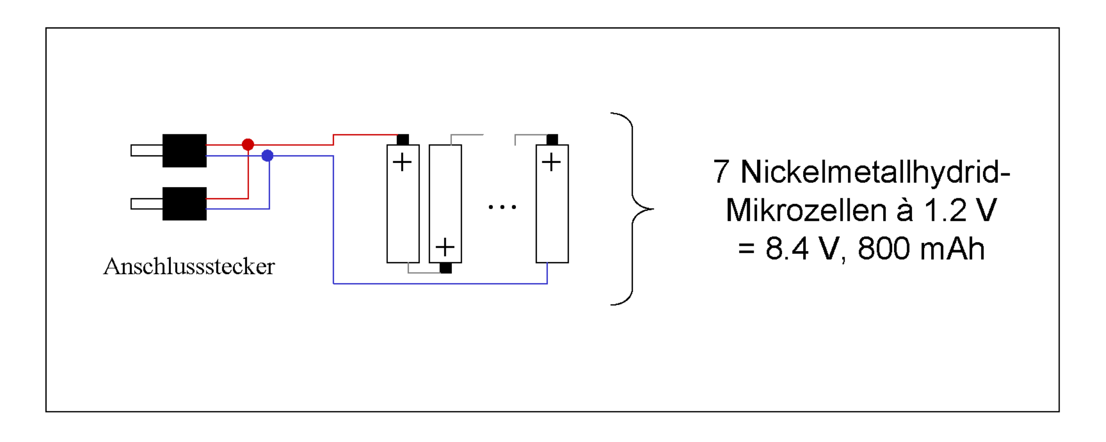


Abbildung 4.19: Kameraakku und Ladeadapter

Die Kamerahalterung und der Akku werden unter zu Hilfenahme beidseitig beschichtetem Klebebandes auf den Roboter montiert. Diese Technik hat sich bewährt, da eine solche Klebeverbindung erstaunlich stabil ist und das Klebeband später rückstandslos entfernt werden kann.

4.4.2.2 i-Glasses

Die **„i-Glasses“** (Abbildung 4.21) ist ein sogenanntes „Head Mounted Virtual Reality Display“. Das Display wird, mittels einer Kopfhalterung, direkt vor den Augen des Betrachters gehalten und basiert auf zwei kleinen TFT-Bildschirmen. Die i-Glasses wird an den Monitorausgang der Grafikkarte angeschlossen und



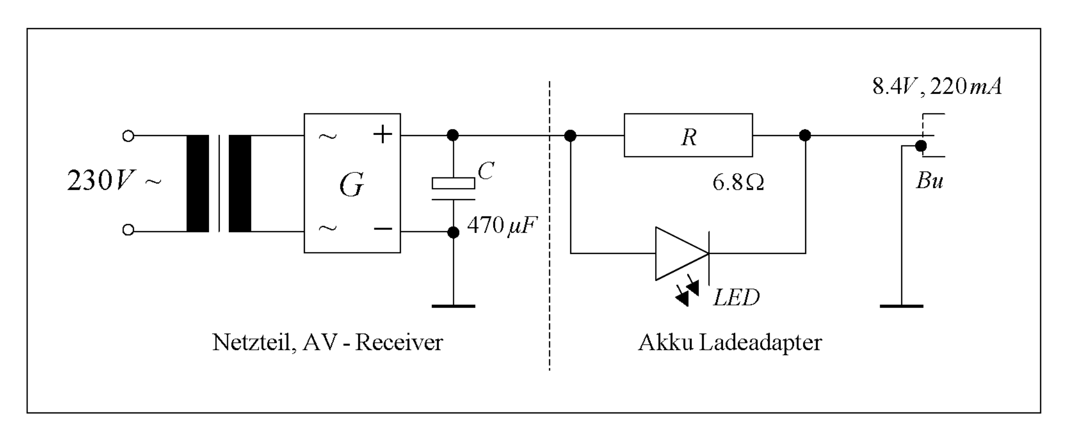


Abbildung 4.20: Schaltskizze - Kameraakku und Ladeadapter

erhält von dieser ein, im Pageflipping-Verfahren (Abschnitt 2.3.4) kodiertes, stereoskopisches Videosignal. Auf der Vorderseite der i-Glasses ist deutlich der aufgeklebte Infrarot-Reflexionspunkt zu sehen. Er ermöglicht dem Headtracking-System die Kopfbewegung des Trägers zu verfolgen.



Abbildung 4.21: i-Glasses, Head Mounted VR-Display

4.4.3 Anwendung: „Stereo View“

Das Programm **„Stereo View“** ist die, im Rahmen dieser Diplomarbeit, entwickelte PC-Anwendung. Sie stellt den programmiertechnischen Teil der vorliegenden Arbeit dar und vereinigt ein Stereokamera-System mit einem Fernsteuer-System für den AIBO ERS-7 Roboter.

Im folgenden soll ein Überblick über die Programmkomponenten und deren Zusammenwirken dargelegt werden.

4.4.3.1 Klassendiagramm

4.4.3.2 *WinMain()*

Die Funktion ***WinMain()*** ist die „Einsprungfunktion“ eines jeden Windows-Programms. Im Falle der Anwendung „Stereo View“ ist sie gleichzeitig ein „Steuermodul“, welches die anderen Programmmodule koordiniert.

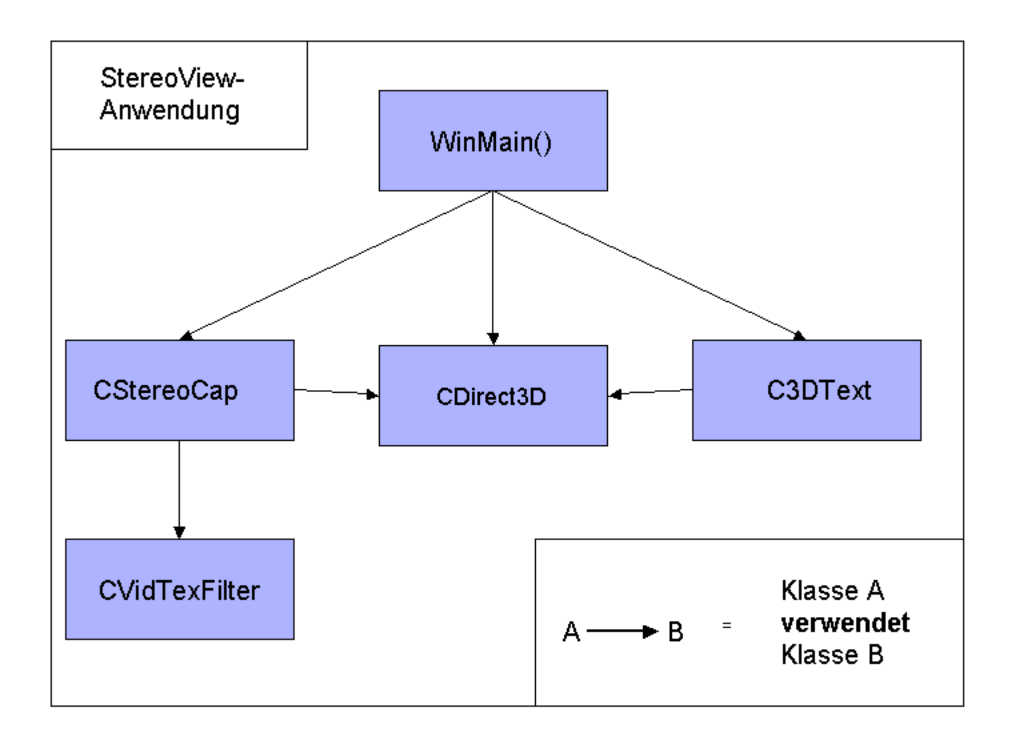


Abbildung 4.22: Klassendiagramm der „Stereo View“-Anwendung

Durch Aufruf der Funktion *WinMain()* wird das Anwendungsfenster erzeugt und nacheinander alle anderen Programmmodule initialisiert und auf ihren Betrieb vorbereitet.

In *WinMain()* befindet sich auch die Windows-Nachrichtenschleife des Programms. Dort erfolgen Reaktionen auf eventuelle Benutzereingaben und die Koordination des Programmablaufs.

Nachdem der Benutzer ein Signal zum Beenden des Programms gegeben hat, werden alle Programmmodule heruntergefahren und das Programm beendet.

4.4.3.3 *CStereoCap*

Die Klasse ***CStereoCap*** implementiert die Programmfunktionen zur Aufnahme, Verarbeitung und Wiedergabe der, von dem an den PC angeschlossenen Doppelkamera-System, eingehenden Bilddaten.

4.4.3.4 *CVidTexFilter*

Der Klasse ***CVidTexFilter*** nimmt eine besondere Rolle ein. Es handelt sich bei dieser Klasse um einen, eigens für das Programm „Stereo View“, entwickelten Directshow-Filter zur Verarbeitung von Videodaten.

Zwei Instanzen dieses Filter (eine für jede Kamera) werden durch ein Objekt der Klasse *CStereoCap* in den Filter-Pool der Directshow-Schnittstelle eingebunden. Jeder der beiden Filter wird in einem eigenen Thread gestartet. Die eingehenden Videostreams können so parallel zum restlichen Programm verarbeitet werden.

4.4.3.5 *CDirect3D*

Die Klasse ***CDirect3D*** vereinfacht die Initialisierung und den Zugriff auf die Direct X-Grafikschnittstelle „Direct-3D“.Sie wird zur Ausgabe der stereoskopischen Bilddaten benötigt.

4.4.3.6 *C3DText*

Die Klasse ***C3DText*** wurde entwickelt um während des Betriebs der Direct-3D-Grafikschnittstelle auf dem stereoskopischen Display Textdaten anzeigen zu können.

Die Verwendung der Windows-GDI[8] Funktionen zur Textdarstellung ist bei Echtzeit-Grafikanwendungen unter Direct-3D nicht möglich. Graphikfunktionen des GDI werden nicht direkt durch den Prozessor des Graphikadapters verarbeitet und senken in der Regel die Bildwiederholrate auf inakzeptable Werte.

4.4.4 Verarbeitung der Videodaten

Die Software des Stereokamerasystems muss zunächst die, vom Kamerasystem eingehenden Videodaten aufnehmen und sie in ein Format bringen, dass über die Direct-3D-Grafikschnittstelle dargestellt werden kann. Das dazu notwendige Vorgehen wird in den folgenden Abschnitten beschrieben.

[8]GDI: Graphics Device Interface.

4.4.4.1 Filtergraphen

Videodaten werden unter Directshow in Form von Datenströmen, sogenannten **Videostreams**, abgebildet.

Der gesamte Weg, den ein Datenstrom unter Directshow beschreitet, wird durch sogenannte **Directshow-Filter** beschrieben. Ein Filter besitzt normalerweise einen Eingabe-Pin und einen, unter Umständen auch mehrere, Ausgabe-Pin/-s. **Pins** sind „Anschlüsse" über die ein Datenstrom in einen Filter eingeleitet bzw. aus ihm ausgeleitet wird. Mehrere Filter können über ihre Ein- und Ausgabe-Pins miteinander verbunden werden. Einen Verbund mehrerer Filter nennt man einen **Filtergraphen**.

Es gibt drei Kategorien von Filtern:

1. Eingabe-Filter
2. Verarbeitungs-Filter
3. Ausgabe-Filter

Ein **Video-Eingabefilter** hat keinen Eingabepin. Seine Eingabe bezieht er stattdessen aus einer **Datenquelle**. Datenquellen können unter anderem Videodateien oder Video-Eingabegeräte sein. Eingabefilter, die Video-Eingabegeräte, beispielsweise eine USB-Kamera, ansprechen bezeichnet man als **Video-Capture-Filter**.

Verarbeitungs-Filter haben einen Eingabe-Pin über den ein Datenstrom in den Filter eingeleitet wird. Innerhalb des Filters wird der eingehende Datenstrom verarbeitet. Über einen, unter Umständen auch mehrere[9], Ausgabe-Pin/-s werden die geänderten Daten aus dem Filter hinaus geleitet. Typische Verarbeitungs-Filter in Video-Datenströmen sind z.b. Farbkonverter und Video En- bzw. Decoder.

Die dritte Filterkategorie sind die **Video-Ausgabefilter**. Sie haben einen Eingabe-Pin, jedoch keinen Ausgabe-Pin. Die Ausgabe erfolgt stattdessen in eine **Datensenke**. Eine typische Datensenke ist ein **Videorenderer**, welcher den

[9] Directshow-Filter besitzen häufig mehrere Ausgabe-Pins, wenn sie einen zusammengesetzten Datenstrom in seine Bestandteile zerlegen (Demultiplexer) oder der ausgeleitete Datenstrom gleichzeitig in unterschiedlichen Formaten bereit gestellt und weiterverarbeitet werden soll.

Videostream auf dem Computerbildschirm darstellt. In der Anwendung „Stereo View“ wird als Ausgabe-Filter ein spezieller Konverter („***CVidTexFilter***“) eingesetzt, der die Daten des Videostreams für die Darstellung auf einem stereoskopischen Display aufbereitet. [2]

Abbildung 4.23 zeigt die schematische Darstellung der, in der Anwendung „Stereo View“ eingesetzten Filtergraphen.

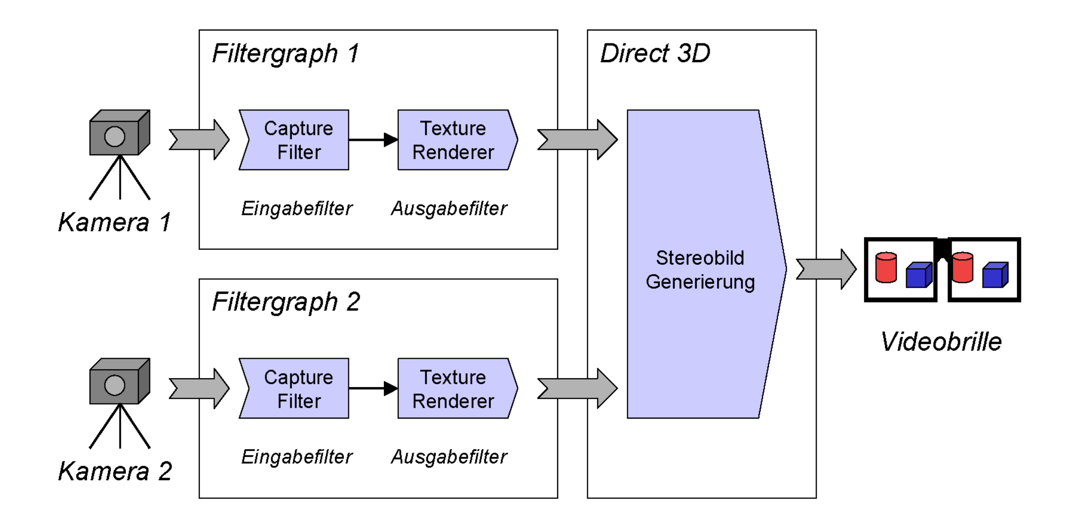


Abbildung 4.23: Filtergraphen der „Stereo View“-Anwendung

4.4.4.2 Video-Capture-Filter

Video-Capture-Filter sind Eingabe-Filter. Innerhalb einer auf Directshow basierenden Anwendung repräsentieren sie ein Video-Eingabegerät, beispielsweise eine an den Computer angeschlossene Kamera oder eine Video-Capture-Karte.

Die Anwendung „Stereo View“ verwendet zwei Capture-Filter als Repräsentanten der beiden Kameras des Doppelkamera-Systems.

Selektion eines Video-Eingabe-Gerätes

Um einen Capture-Filter als Datenquelle eines Directshow Filtergraphen verwenden zu können, muss dieser zunächst mit einem Video-Eingabe-Gerät verknüpft werden.

Das Programm „Stereoview“ weiß grundsätzlich nicht, welche der an dem PC angeschlossenen Video-Geräte den linken und rechten Bildkanal des Stereokamera-Systems repräsentieren. Aus diesem Grund muss der Benutzer beim Start der Anwendung die beiden Videogeräte in einem Konfigurationsdialog (Abbildung 4.24) manuell auswählen.

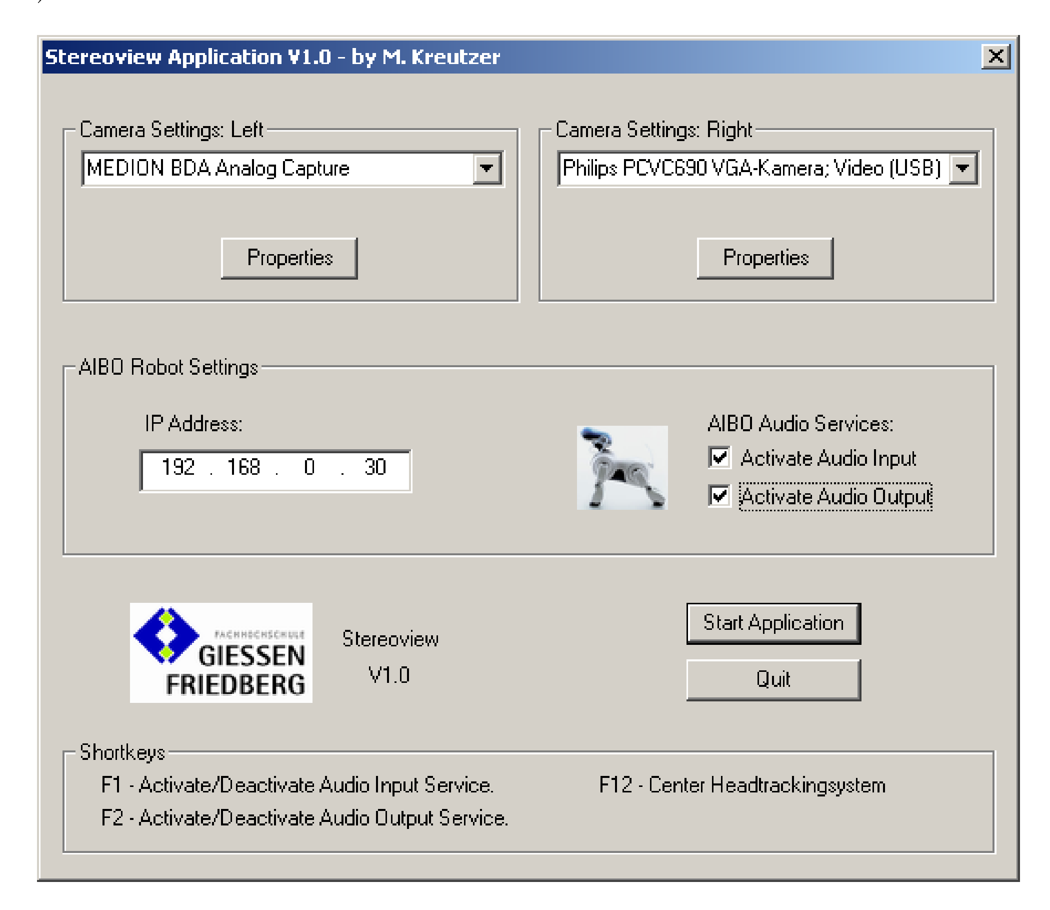


Abbildung 4.24: Konfigurationsdialog der Stereoview Anwendung

Dazu werden alle an das System angeschlossenen Video-Capture-Geräte durch die Methode *CStereoCap::EnumerateDevices()* der *CStereoCap*-Klasse aufgelistet.

Die, an einen Computer angeschlossenen, Multimediageräte sind in Geräte-Kategorien aufgeteilt. Hier sollen alle Geräte der Kategorie „Video-Input-Devices“ aufgelistet werden. Zur Aufzählung wird ein entsprechender **„System-Device-Enumerator“** generiert.

Der „System-Device-Enumerator“ ist ein Hilfsobjekt. Er liefert für jedes Videogerät ein Beschreibungsobjekt des Typs ***„IMoniker“***. Das Beschreibungsobjekt

ermöglicht der Anwendung Informationen über die einzelnen Geräte abzufragen, insbesondere den für Menschen lesbaren Namen des Gerätes. [13]

Die Gerätenamen werden von der „Stereo View“-Anwendung ausgelesen und dem Benutzer zur Auswahl eines Gerätes in einer Dialogbox dargestellt.

Erstellung eines Capture-Filters

Nachdem der Benutzer die gewünschten Video-Eingabe-Geräte ausgewählt hat, erstellt die Anwendung die zugehörigen Video-Capture-Filter.

Die Erstellung eines Capture-Filters findet, programmiertechnisch, durch das Verknüpfen eines Geräte-Beschreibungsobjektes des Typs *IMoniker* mit einem generischen Directshow-Filter-Objekt des Typs ***IBaseFilter*** statt. Abbildung 4.25 zeigt ein entsprechendes Code-Beispiel.

```
// Create an empty pointer to a
// IBaseFilter - Interface
IBaseFilter *pCap = NULL;

// Get the Device-Interface to the
// Capture-Filter
hr = pMoniker->BindToObject(0, 0,
          IDD_IBaseFilter, (void**)&pCap);

if(FAILED(hr)) {
   // An error occured!
}
```

Abbildung 4.25: Listing-Erstellung eines Video-Capture-Filters

Als erstes wird ein „leerer“ Zeiger auf ein Basis-Filter-Interface (***IBaseFilter***) erstellt. Die Methode *IMoniker::BindToObject(..)* erstellt einen neuen Video-Capture-Filter und füllt die übergebene Zeigervariable mit der Adresse des Filter-Interfaces.

4.4.4.3 Video Rendering Filter

Video-Renderer sind Directshow-Ausgabe-Filter. Sie wandeln einen Videostream in ein, auf dem Computerbildschirm darstellbares, Format um oder schreiben den Stream in eine Datei.

Der in Directshow implementierte Standard-Video-Renderer öffnet zur Darstellung eines Videostreams ein eigenes Fenster, in welchem das Video angezeigt wird.

Im Bezug auf die „Stereo View"-Anwendung ist der Standard-Video-Renderer ungeeignet. Die Anwendung soll die Bilder der angeschlossenen Kameras schließlich nicht getrennt in zwei Bildschirm-Fenstern anzeigen. Die Anwendung soll die Videostreams „mischen" und auf einem stereoskopischen Display anzeigen.

Die einzige Möglichkeit das vorliegende Problem zu lösen war die Implementierung eines neuen Video-Ausgabe-Filters, dem **„Texture-Rendering-Filter"** (***Klasse CVidTexFilter***).

Ziel des Texture-Rendering-Filters

Das Stereo-Kamerasystem liefert nach der Digitalisierung der Kamerasignale zwei Video-Daten-Streams. Die Bilder der Videostreams sollen über die Grafikschnittstelle Direct- 3D zusammengeführt und stereoskopisch dargestellt werden.

Um dies zu realisieren, müssen die Bilder der Videostreams in ein für Direct-3D verständliches Format gebracht werden.

2D-Bilddaten, wie die Einzelbilder eines Videostreams es sind, werden in der Welt der 3D-Grafik als **Texturen** bezeichnet. Um einen Videostream unter Verwendung von Direct-3D anzeigen zu können, müssen die Einzelbilder des Streams in Echtzeit auf Direct-3D Texturen übertragen werden.

An dieser Stelle entsteht folgendes Problem: Das Datenformat der Einzelbilder eines Videostreams und das Format der Direct-3D Texturen unterscheiden sich erheblich; sowohl in der Ausrichtung der Bildzeilen, als auch durch das Farbformat der einzelnen Bildpixel. Die Abbildungen 4.26 und 4.27 zeigen schematisch die Anordnung der Bilddaten im Speicher des Rechners bei einem Video-Datenformat bzw. einem Direct-3D Texturdatenformat.

Bilder eines Videostreams werden von unten nach oben aufgebaut, das heißt die erste Bildzeile im Datenarray der zugehörigen Bitmap repräsentiert die unterste Zeile des tatsächlichen Bildes. Die zweite Zeile im Array die vorletzte Zeile des Bildes usw. Die Bildausrichtung auf Direct-3D Texturen ist genau umgekehrt. Das Bild wird von oben nach unten aufgebaut. Die erste Zeile im Bilddatenarray

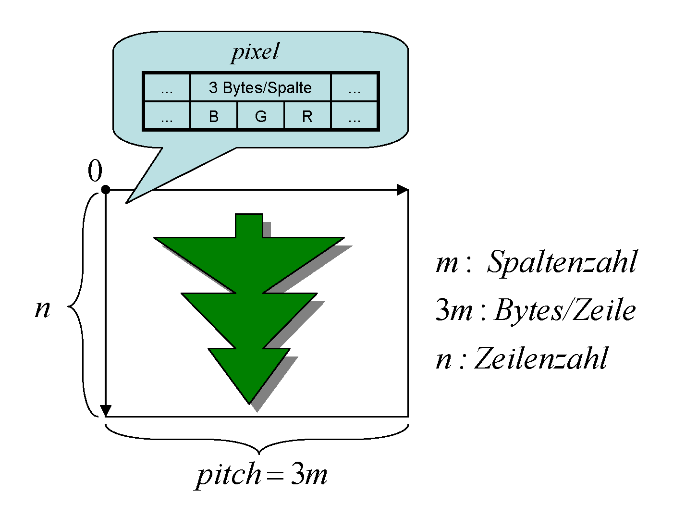


Abbildung 4.26: Videodatenformat

entspricht genau der ersten Zeile des tatsächlichen Bildes. Darüber hinaus kann jede Bildzeile der Texturdaten an ihrem Zeilenende Steuerdaten für die Grafikkarte enthalten.

Zusätzlich zur Bildausrichtung unterscheidet sich die Farbkodierung der Pixel der beiden Formate. Ein Bild des Videostreams besitzt das Farbformat „B8G8R8"(24 Bit). Ein Bildpixel besteht demnach aus 3 Bytes (3 x 8 Bit), wobei das erste Byte die Blaukomponente, das zweite die Grünkomponente und das dritte Byte die Rotkomponente des Farbtons speichert. Ein typisches Pixel-Farbformat einer Direct-3D-Textur, die Verwendung einer modernen Grafikkarte vorausgesetzt, ist dahingegen „X8R8G8B8" (32-Bit). Das erste Byte eines solchen Pixels (X8) ist unbenutzt bzw. speichert die Transparenz eines Pixels (Alpha-Kanal). Die folgenden Bytes enthalten in dieser Reheinfolge die Farbkomponenten Rot, Grün und Blau. Tabelle 4.4 zeigt eine Gegenüberstellung eines Video- und eines Textur-Farbformats.

Tabelle 4.4: Vergleich: Farbformate- Video und D3D-Textur

Farbformat	Byte 1	Byte 2	Byte 3	Byte 4
B8G8R8	-	Blau	Grün	Rot
X8R8G8B8	Alpha	Rot	Grün	Blau

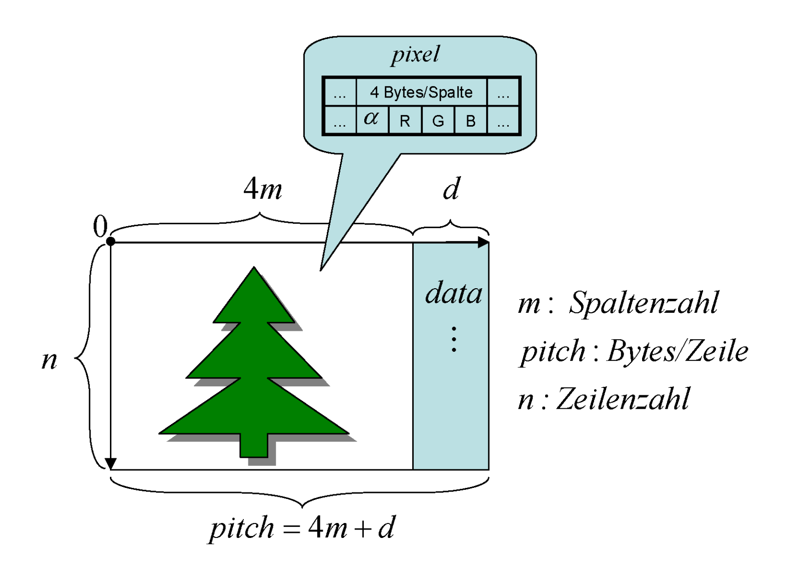


Abbildung 4.27: Texturdatenformat

Die einzige Lösung des oben beschriebenen Problems ist die byteweise Umkodierung aller Einzelbilder des Videostreams. Unter Umständen ist gar eine bitweise Umkodierung der Pixel erforderlich, nämlich dann, wenn die, in Direct-3D eingestellte, Farbtiefe nur 16 Bit statt 32 Bit beträgt. Dieser Fall wird von dem, für die „Stereo View"-Anwendung implementierten Video-Ausgabe-Filter (*CVidTexFilter*) ebenfalls abgefangen, soll hier aber nicht näher besprochen werden.

Implementierung des Video-Texture-Renderers

Microsoft stellt dem Anwendungs-Entwickler eine Reihe von Hilfsklassen zur Verfügung, welche die Entwicklung eigener Directshow-Filter vereinfachen. Zur Entwicklung eines neuen Video-Ausgabe-Filters wird dessen Programmklasse von der Directshow Basisklasse *CBaseVideoRenderer* abgeleitet.

Die Basisklasse implementiert alle grundlegenden Funktionen eines Video-Rendering-Filters. Der Programmierer überschreibt lediglich die Methoden, die für die Verarbeitung der Videodaten zuständig sind und passt diese so dem neuen Verwendungszweck an

Neu implementiert werden müssen folgende Methoden der Basis-Klasse *CBaseVideoRenderer*:

***CheckMediaType(...)*:** Methode zur Abfrage der, von einem Filter unterstützten, Datenformate.

***SetMediaType(...)*:** Methode zum Setzen eines gewählten Datenformats.

***DoRenderSample(...)*:** Methode zur Verarbeitung eines Einzelbildes (Sample) des Videostreams.

Die Methode *CheckMediaType(...)* wird von Directshow während des Aufbauprozesses eines Filtergrafen verwendet um zu überprüfen, ob der Filter ein angebotenes **Datenformat** unterstützt. Der Begriff „Datenformat“ meint im Falle eines Video-Ausgabe-Filters in erster Linie das Farbformat des Videostreams. Der Methode wird ein Zeiger auf eine Datenstruktur des Typs *CMediaType* übergeben. Durch Auslesen der Datenstruktur ermittelt der Filter das angebotene Datenformat und prüft, ob es durch ihn verarbeitet werden könnte. Ist das der Fall, muss die, durch den Filter implementierte Methode *CheckMediaType(...)* den Wert des Define-Makros *S_ OK* zurückliefern. Kann ein Datenformat nicht verarbeitet werden, wird dies durch Rückgabe des Makros *E_INVALID_PARAMETERS* signalisiert.

Die Implementierung dieser Methode in der von *CBaseVideoRenderer* abgeleiteten Filter-Klasse *CVidTexFilter* akzeptiert nur Video-Formate mit dem Farbformat „RGB-24“. Dieses entspricht dem in Tabelle 4.4 angegebenen Farbformat „B8G8R8“ für Videostreams. In der Regel wird dieses Format von allen, dem Video-Ausgabe-Filter vorgeschalteten, Video-Decoder-Filtern bzw. Video-Capture-Filtern unterstützt. Der Programmieraufwand zur Konvertierung eines Videostreams in ein Direct-3D-Texturformat kann so maßgeblich reduziert werden.

Es ist möglich, dass ein Filter mehrere, durch die Methode *CheckMediaType(...)* abgefragte Formate unterstützt. Directshow entscheidet sich in diesem Fall für das Datenformat, welches den Aufbau des effizientesten Filtergrafen ermöglicht. Auf diese Weise kann auf den Einsatz überflüssiger Konvertierungs-Filter verzichtet werden.

Hat Directshow sich für ein Datenformat entschieden, so wird dieses durch den Aufruf der Filter-Klassen-Methode *SetMediaType(...)* festgeschrieben. Nach Auf-

ruf der Methode und einer positiven Rückmeldung des Filters ist davon auszugehen, dass der Filter korrekt von Directshow in den Filtergrafen eingesetzt wurde.

Sobald der Filtergraph gestartet wird, ruft Directshow für jedes Einzelbild des Videostreams (Sample) die Methode *DoRenderSample(...)* des neuen Ausgabe-Filters auf. Im Falle der, für die „Stereo View“-Anwendung entwickelten, *CVidTexFilter*-Klasse übernimmt diese Methode die Dekodierung der Einzelbilder und die Neukodierung in das Datenformat einer Direct-3D-Textur.

Der Methode *DoRenderSample(...)* wird von Directshow ein Zeiger auf ein Objekt der Klasse *„IMediaSample“* übergeben. Durch Aufruf der Methode *GetPointer(...)* des *IMediaSample*-Objekts erhält der Programmierer einen Byte-Zeiger, der auf die Bilddaten des zu verarbeitenden Einzelbildes (Sample) zeigt.

Ausgabe auf eine Direct-3D-Textur

Um das Ergebnis der Datenformat-Konvertierung in einer Direct-3D-Textur abzulegen, wird des weiteren ein Zeiger auf die Bilddaten der Textur benötigt. Diesen erhält man durch Aufruf der Methode *LockRect(...)* eines zuvor angelegten Textur-Objekts. Die Methode erlaubt den exklusiven Zugriff auf ein so gesperrtes Rechteck der Textur. Eine, von der Methode *LockRect(...)* mit Daten gefüllte Struktur, die ***D3DLOCKED_RECT*-Struktur**, enthält eine Zeigervariable mit der Bezeichnung *pBits*. Der Zeiger *pBits* zeigt auf den Speicherbereich, indem die Bilddaten der Textur abgelegt sind und ermöglicht direkten Zugriff auf diese Daten.

Das Sperren einer Textur durch Aufruf der Methode *LockRect(...)* sorgt dafür, dass die, normalerweise im Bildspeicher der Grafikkarte abgelegte, Textur in den Hauptspeicher des Rechners kopiert wird. Erst dort können die Bilddaten durch den Hauptprozessor bearbeitet werden. Solange die Textur gesperrt ist, kann die Grafikkarte nicht auf sie zugreifen. Bevor die Textur gezeichnet werden kann muss die Methode *UnLock()* des Textur-Objekts aufgerufen werden, um die Textur zu entsperren.

Die Erzeugung und allgemeine Benutzung von Direct-3D-Texturen wird in der Dokumentation der Direct X-Schnitstelle [13] im Unterkapitel „Direct X Graphics“ exakt beschrieben und kann dort nachgeschlagen werden.

Konvertierung des Video-Datenformats

Nachdem nun der Zugriff auf das zu verarbeitende Bild des Videostreams, sowie der Zugriff auf die Textur-Bilddaten gewährleistet ist, kann mit der Konvertierung der Bilddaten fortgefahren werden.

Wie bereits beschrieben, werden die Daten eines Videobildes zeilenweise, beginnend mit der untersten Bildzeile im Speicher abgelegt (Abbildung 4.26). Das Videobild steht also, im übertragenen Sinne, auf dem Kopf. Die Bilddaten einer Textur sind ebenfalls zeilenweise, jedoch beginnend mit der obersten Zeile des Bildes, im Speicher angeordnet. Jede im Speicher abgelegte Texturzeile kann an ihrem Ende zusätzliche Steuerinformationen der Grafikkarte enthalten (Abbildung 4.27). Der Wert, um den ein Zeiger auf die Bilddaten einer Textur erhöht werden muss, um in die nächste Zeile zu gelangen kann daher nicht durch Aufsummierung der bildtragenden Datenbytes errechnet werden. Man benötigt eine weitere Angabe, den sogenannten **Zeilenabstand (Pitch)**. Dieser Wert besagt, um wie viele Byte-Schritte ein Zeiger erhöht werden muss, um von einer Zeile der Textur in die nächste zu gelangen. Der Zeilenabstand kann aus der Variablen *Pitch* der *D3DLOCKED_RECT*-Struktur ausgelesen werden.

Neben der unterschiedlichen Ausrichtung unterscheiden sich die Bilddaten eines Videostreams von denen der Direct-3D-Texturen durch ihre Farbformate. Das „B8G8R8"-Farbformat der Videobilder muss in ein Textur-Farbformat, beispielsweise das „X8R8G8B8"-Format, umgerechnet werden.

Beispielalgorithmus zur Bilddatenkonvertierung

Abbildung 4.28 zeigt eine beispielhafte Implementierung des Konvertierungsalgorithmus.

Zu Beginn des Beispielalgorithmus wird zunächst jeweils ein Zeiger auf die Daten des Videobildes und ein Zeiger auf die Daten der Textur ausgelesen. Das Speicherabbild des Videobildes ist, gegenüber dem Original, an der horizontalen Achse gespiegelt. Das Bild steht auf dem „Kopf". Die erste Zeile des Datenblocks entspricht somit der letzten Zeile des Bildes. Um die Ausrichtung des Bildes zu korrigieren, muss der Konvertierungsvorgang mit der letzten Zeile des Datenblocks beginnen und mit der ersten Zeile enden. Die Operation „$pVideo\ +=$

```
// Get pointer to video-sample-data
// and set it to the first row
// of the picture (last row in data-block)

BYTE *pVideo;
pSample->GetPointer( &pVideo );
pVideo += dwVideoWidth*(dwVideoHeight-1)*3;

// Get pointer to texture-data
BYTE* pTexture = (BYTE*)(LockedRect.pBits);

// Get pitch and recalculate to pixel/row
// (4 Bytes per pixel)
DWORD dwPitch = LockedRect.Pitch / 4;

// Convert video-data to texture-data
for(DWORD y = 0; y < dwVideoHeigth; y++)
{
  for(DWORD x = 0; x < dwVideoWidth; x++)
  {
    *(pTexture++) = 255;          // unused/alpha
    *(pTexture++) = pVideo[2];    // Red
    *(pTexture++) = pVideo[1];    // Green
    *(pTexture++) = pVideo[0];    // Blue
    pVideo += 3;
  }

  // skip additional data and go to the next
  // row of the texture.
  pTexture += (dwPitch - dwVideoWidth)*4;

  // set *pVideo to the next row of the
  // picture (one row back in data-block)
  pVideo -= dwVideoWidth * 6;
}
```

Abbildung 4.28: Listing-Konvertierungsalgorithmus

*dwVideoWidth * (dwVideoHeight-1) * 3*“ des Beispielalgorithmus setzt den Byte-Zeiger *pVideo* auf die letzte Zeile des Datenblocks und somit auf die erste Zeile des tatsächlichen Videobildes.

Als nächstes wird der Textur-Zeilenabstand (Pitch) aus der *D3DLOCKED_RECT*-Struktur ausgelesen und durch 4 dividiert. Mittels der Division durch 4 wird der Zeilenabstand von der Einheit „Bytes/Zeile“ in die Einheit „Bildpixel/Zeile“ umgerechnet.

Innerhalb der zweifach verschachtelten *For*-Schleife werden die Farbkomponenten eines jeden Pixels des Videobildes ausgelesen, konvertiert und in den Bildpuffer der Textur geschrieben.

Nach der Bearbeitung einer Bildzeile muss der Zeiger *pTexture*, der auf die Texturdaten zeigt, an den Anfang der nächsten Zeile bewegt werden. Da am Ende der aktuellen Zeile noch weitere Daten (Steuerinformationen der Grafikkarte) stehen können muss der auf den Zeiger zu addierende Wert durch die Rechenoperation „*(dwPitch - dwVideoWidth)*4*“ bestimmt werden. Die Multiplikation mit 4 führt zur Umrechnung der Einheit von „Pixel/Zeile“ nach „Bytes/Zeile“ (*pTexture* ist ein BYTE-Zeiger).

Der Zeiger *pVideo*, der auf die Daten des Videobildes zeigt steht nach Abarbeitung der inneren *For*-Schleife am unmittelbaren Ende der Bildzeile und muss um zwei Bildzeilen zurückgesetzt werden. Dies entspricht der nächsten Zeile des tatsächlichen Videobildes und geschieht durch die Rechenoperation „*pVideo -= dwVideoWidth * 6*“. Die Variable *dwVideoWidth* gibt die Breite des Videobildes in Pixel an. Ein Pixel des Videobildes besteht aus drei Bytes. Die Operation „*dwVideoWidth * 6*“ liefert als Ergebnis die Länge von genau zwei Bildzeilen in Bytes und wird von dem Zeiger *pVideo* subtrahiert.

Eigenschaften der Filterklasse „*CVidTexFilter*“

Die Filter-Klasse *CVidTexFilter* implementiert neben dem in Abbildung 4.28 dargestellten Konvertierungs-Algorithmus eine Anzahl weiterer Algorithmen zur Unterstützung unterschiedlicher Textur-Farbformate. Das von der Filter-Klasse gewählte Format für die Farben der Zieltextur hängt von den Fähigkeiten des Grafikadapters und den Einstellungen des Direct-3D-Display-Formats ab. Die

Auswahl des Texturformats und der zugehörigen Algorithmen geschieht automatisch.

Der Filter nutzt zur Konvertierung des Videostreams die Ressourcen des Hauptprozessors. Dieser trägt, insbesondere bei der parallelen Konvertierung mehrerer Streams (Doppelkamera-System), eine beträchtliche Arbeitslast. Zum aktuellen Zeitpunkt gibt es keine Möglichkeit den Videostream über den, eigentlich für solche Aufgaben ausgelegten, Prozessor der Grafikkarte zu konvertieren. Das etwas exotisch anmutende Datenformat der Videostreams ist offensichtlich ein Derivat der historischen Entwicklung des Betriebssystems „Microsoft Windows". Es wäre wünschenswert, dass der Betriebssystemhersteller „Microsoft" zusammen mit den Herstellern von Video- und Grafikhardware eine akzeptable Lösung[10] für das beschriebene Konvertierungsproblem entwickelt.

4.4.5 Reproduktion der Videodaten

Das Ziel der Videodaten-Reproduktion ist die Wiedergabe der eingehenden Videostreams über ein stereoskopiefähiges Display. In dieser Arbeit wurde zur stereoskopischen Ausgabe ein **„Virtual Reality Helm"** (I-Glasses, Abbildung 2.13) verwendet. Das Gerät besitzt zwei kleine TFT-Displays, die direkt vor den Augen des Betrachters platziert sind. Jedes TFT-Display deckt das Gesichtsfeld eines Auges ab.

Das Bild der linken Kamera des Doppelkamerasystems soll auf dem linken TFT-Display abgebildet werden. Analog dazu muss das Bild der rechten Kamera auf dem rechten TFT-Display der „Virtual Reality Brille" abgebildet werden.

Das alles klingt zunächst sehr trivial. Aufgrund der Tatsache, dass „herkömmliche" Computersysteme weder von der Hard-, noch von der Softwareseite für das Darstellen stereoskopischer Bilddaten ausgelegt sind, ist das dazu notwendige Vorgehen aber durchaus komplex.

Um das Problem der Reproduktion stereoskopischer Videodaten im Rahmen dieser Diplomarbeit zu lösen mussten drei unterschiedliche Programmier-Ansätze getestet werden. Der dritte Ansatz führte jedoch zu einem ausgezeichneten Ergebnis.

[10] In Form angepasster Schnittstellen und Hardware.

Das Unterkapitel befasst sich in erster Linie mit der technischen Realisierung dieses erfolgreichen Ansatzes: Der Reproduktion stereoskopischer Bilddaten über den Umweg der 3D-Computergrafik.

Die ersten, fehlgeschlagenen, Programmieransätze waren grundsätzlich vielversprechend und könnten für die Lösung ähnlicher Problemstellungen in der Zukunft durchaus attraktiv sein. Aufgrund dessen soll auf eine kurze Beschreibung nicht gänzlich verzichtet werden.

4.4.5.1 Zwei fehlgeschlagene Programmieransätze zur Darstellung stereoskopischer Bilddaten

Ansatz 1: Darstellung stereoskopischer Bilddaten mit dem „NVIDIA Stereo Blit SDK“

Die Darstellung stereoskopischer Bilddaten ist in der Direct-3D-Grafikschnittstelle (aktuelle Version 9) nicht vorgesehen und wurde dementsprechend dort nicht implementiert. Um dennoch stereoskopische Bilder darstellen zu können, schlägt der Grafikchiphersteller NVIDIA für seine Grafikkarten eine Speziallösung in Form des **„NVIDIA-Stereo-Blit SDKs“** vor.

Es soll hier vorweggenommen werden, dass dieses SDK in Wahrheit nur ein, von NVIDIA inoffiziell herausgegebenes Direct-3D-Beispielprogramm ist. Das Programm führte auf dem Betriebssystem „Windows 2000“ unter allen getesteten Kombinationen von Grafikkarten und Computersystemen zu einem Systemabsturz. Obwohl der Einsatz des NVIDIA-Stereo-Blit SDKs für die Implementierung der hier entwickelten Anwendung „Stereo View“ zu keinen Erfolgen führte, soll das grundsätzliche Verfahren kurz beschrieben werden. Es bestehen gute Chancen, dass die Entwicklung dieses SDKs durch die Firma NVIDIA in naher Zukunft soweit gediehen ist, dass es zur Entwicklung von Stereoskopie-Anwendungen eingesetzt werden kann. Der folgende Abschnitt soll zukünftigen Entwicklern auch die aufwendige Analyse des SDK-Quellcodes ersparen. In dem umfangreichen Direct-3D-Programm finden sich keinerlei Hinweise auf die Art des angewendeten Verfahrens oder die wesentlichen Stellen des Quellcodes. Hier würde ein Programmierer der alten Schule möglicherweise anmerken: „Wozu dokumentieren und kommentieren, alles was getan werden soll steht im Programmcode.“

Die von NVIDIA vorgeschlagene Lösung basiert auf einer Ergänzung des Direct-3D-Texturformats, welche die Definition stereoskopischer Texturen ermöglicht.

Eine stereoskopische Textur besitzt zwei Bildkanäle. Der erste Kanal speichert das Bild, dass für das linke Auge des Betrachters vorgesehen ist. Der zweite Bildkanal das Bild für das rechte Auge. In der Computer-Stereografie gibt es einen häufig eingesetzten „Trick“ um ein stereoskopisches Bild in herkömmlichen Bilddatenformaten, beispielsweise dem Direct-3D-Texturformat, zu speichern. Dazu werden die Einzelbilder des Stereobildes horizontal, nebeneinander angeordnet und in den Bildspeicher einer gewöhnlichen Direct-3D-Textur kopiert. Durch die horizontale Anordnung der beiden Bildkanäle vergrößert sich die Textur auf die doppelte Breite eines Einzelbildes. Direct-3D kann diese Art stereoskopischer Texturen von „gewöhnlichen“ Texturen nicht unterscheiden und lässt daher alle Texturoperationen problemlos zu.

Die Zusammensetzung des zweispaltigen Texturbildes zu einem darstellbaren Stereobild erfolgt erst mehrere Softwareebenen unterhalb der Direct-3D-Schnittstelle, durch den speziellen Stereo-Zusatztreiber der NVIDIA-Grafikkarten. Damit der NVIDIA-Stereo-Treiber die stereoskopischen Texturen von „normalen“ Texturen unterscheiden kann, wird eine spezielle Signatur in die Bilddaten der Stereo-Texturen eingebracht. Diese Signatur ist nichts weiter als eine Konstante des Typs „*LONG*“, also eine sehr große Zahl. Sie wird in der Header-Datei des NVIDIA-Stereo-Blit SDKs definiert.

Anhand der Signatur erkennt der NVIDIA-Stereo-Treiber die stereoskopische Textur, teilt sie vertikal in ihrer Mitte und trennt die Textur so in die zwei Einzelbilder des Stereobildes auf. Diese werden durch die Grafikkarte verarbeitet und über ein stereoskopiefähiges Display an das linke und rechte Auge des Betrachters weitergeleitet.

Dieses Konzept ist grundsätzlich sehr flexibel und ließe gar die Verwendung stereoskopischer Texturen in 3D-Computerspielen zu, wenn die zugehörige Treibersoftware technisch ausgereift wäre.

Ansatz 2: Manuelle Implementierung des Page-Flipping-Verfahrens

Wie im vorigen Abschnitt bereits angedeutet erwies sich das, von NVIDIA angebotene, Software-Entwicklungs-Paket zur Darstellung stereoskopischer Bilddaten, zum aktuellen Zeitpunkt als unbrauchbar.

Ein nächster möglicher Ansatz wäre das direkte Ansprechen der Grafikkarte über ihre Grafiktreiber. Auf diesem Wege könnte ein eigenes Verfahren zur Aufbereitung eines stereoskopischen Videosignals implementiert werden. Im Rahmen dieser Arbeit wurde ein Versuch gestartet das sogenannte **„Page-Flipping“ Verfahren** - hier wird abwechselnd, im Takt der vertikalen Bildsynchronisation, jeweils das Bild für das linke bzw. das rechte Auge angezeigt - manuell zu implementieren. Dieser Versuch lieferte interessante Ergebnisse, scheiterte jedoch letztlich an Synchronisationsproblemen zwischen Anwendung und Grafikkarte. Die mangelnde Synchronisation führte im Takt weniger Minuten immer wieder zur Vertauschung der Bildkanäle des linken bzw. rechten Auges. Da durch die sehr hardwarenahe Programmierung weitere Risiken und hohe Inkompatibilität zu unterschiedlichen Grafikadaptern zu befürchten waren, wurde die Weiterverfolgung dieses Ansatzes eingestellt.

4.4.5.2 Darstellung stereoskopischer Bilder über die 3D-Grafikschnittstelle Direct-3D

Nachdem die beiden ersten Ansätze zur Reproduktion stereoskopischer Bilddaten fehlschlugen, sollte ein ganz neuer Ansatz verfolgt werden.

Viele bekannte Hersteller von Grafikchipsätzen, darunter die Firmen NVIDIA und ATI, liefern für ihre Grafiktreiber **Stereoskopie-Zusatzmodule** zur stereoskopischen Darstellung dreidimensionaler Computergrafik[11].

Das Stereoskopie-Zusatzmodul bildet dazu eine zusätzliche Softwareschicht zwischen der Direct-3D-Grafikschnittstelle und den Treibern der Grafikkarte. Direct-3D unterstützt, wie bereits erwähnt, keine stereoskopischen Darstellungsverfahren. Das Zusatzmodul „zwingt“ die Grafikkarte jedoch dazu, die von der Grafikschnittstelle gelieferte 3D-Modelle aus zwei unterschiedlichen Perspektiven zu

[11]Wir kennen solche texturierten 3D-Vektorgrafiken aus modernen Computerspielen oder CAD-Anwendungen.

rendern. Eine Perspektive liefert das Bild für das linke Auge, während die zweite Perspektive das Bild für das rechte Auge liefert. Der Betrachter nimmt das dargestellte 3D-Modell also so wahr, als würde er es in der „realen“ Welt mit beiden Augen betrachten. Abbildung 4.29 zeigt eine kleine 3D-Szene, bestehend aus den Modellen eines Würfels und eines Zylinders. Die Szene wird aus der Sicht zweier virtueller Kameras aufgenommen und auf die Displays einer Stereo-Brille übertragen. Die Kameras sind - den Maßstab der virtuellen Szene eingerechnet - im „Augenabstand“ angeordnet.

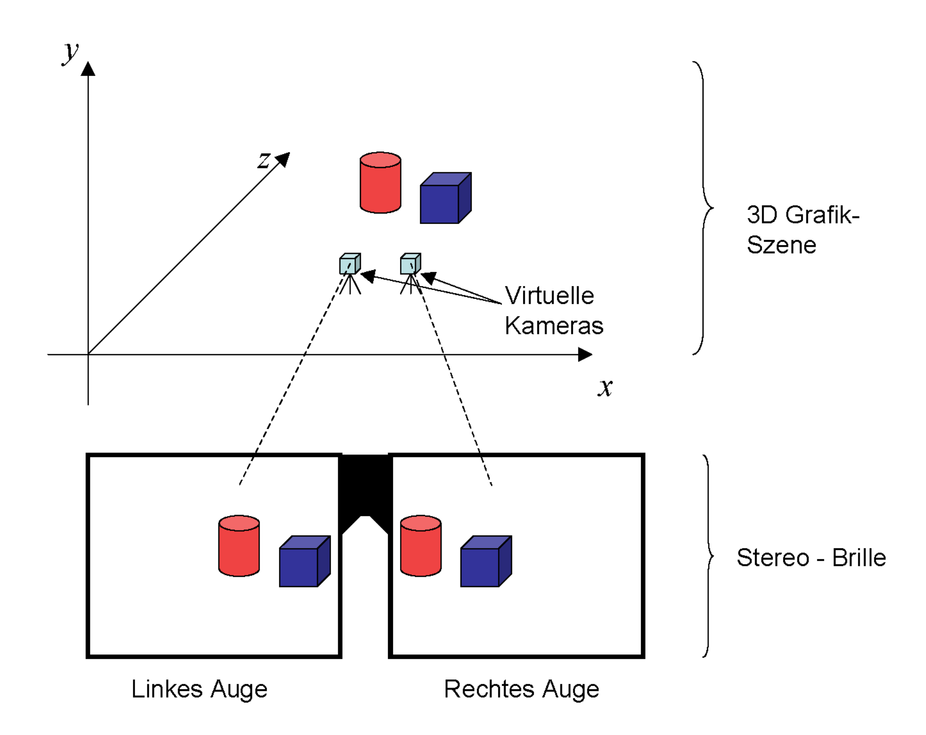


Abbildung 4.29: 3D-Grafikszene in stereoskopischer Darstellung

Nun ist es so, dass das für die „Stereo View“-Anwendung konstruierte Doppelkamera-System kein dreidimensionales Modell der abzubildenden Szene liefert, sondern lediglich zwei, aus unterschiedlichen Perspektiven aufgenommene, zweidimensionale „Abbilder“ der „realen“ Welt. Diese „Abbilder“ können jedoch in eine dreidimensionale Grafikszene projiziert werden und so über das Stereoskopie-Zusatzmodul des Grafikkartenherstellers dargestellt werden.

Abbildung 4.30 zeigt die Konstruktion einer 3D-Grafikszene, auf Grundlage der Videostreams eines Doppelkamera-Systems.

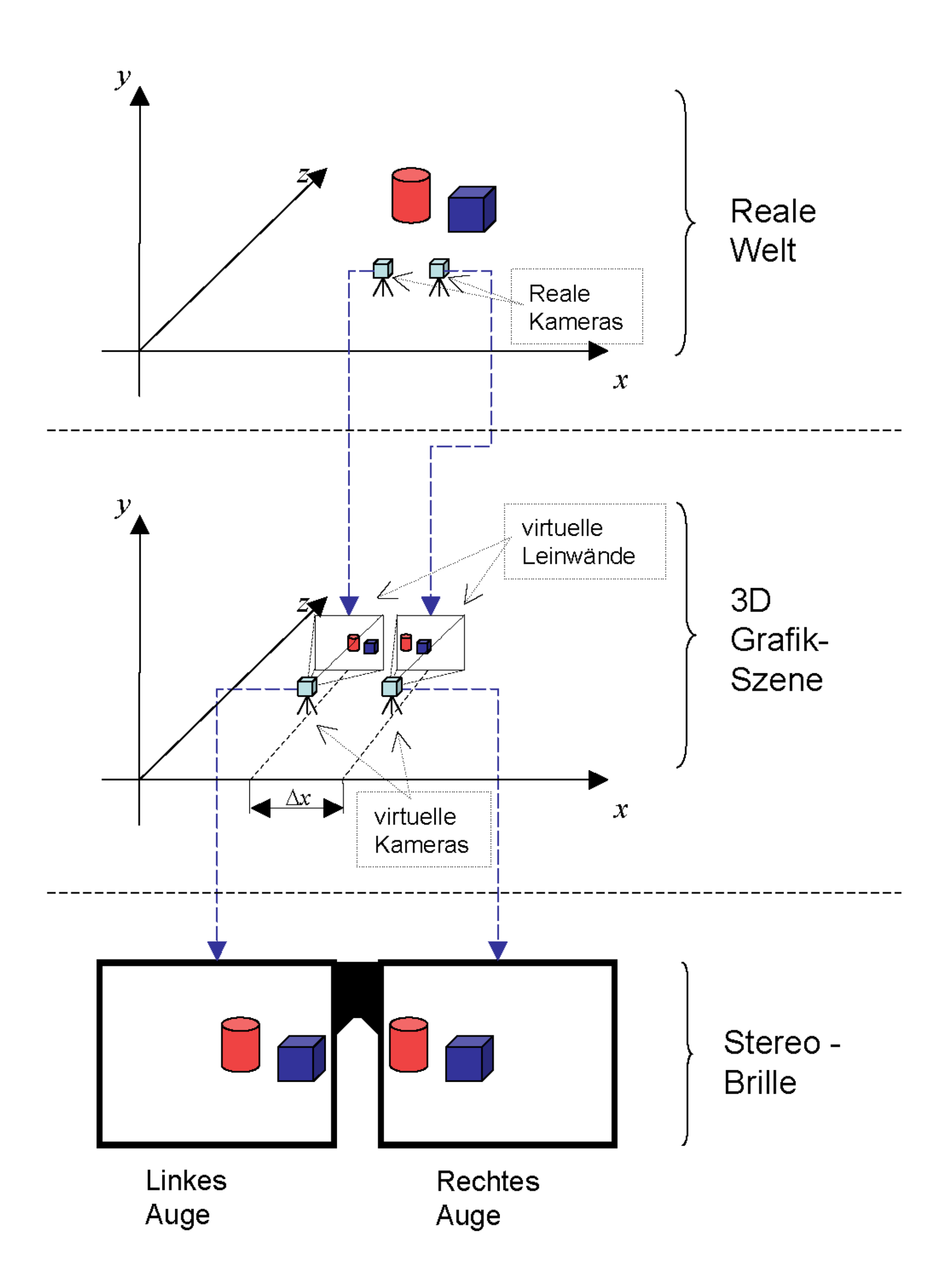


Abbildung 4.30: Konstruktion: 3D-Grafikszene aus den Bildern eines Doppelkamerasystems

Das oberste der drei Teilbilder stellt ein Szenario in der „realen Welt“ dar. Zwei massive Körper, ein Zylinder und ein Würfel, werden durch ein Doppelkamera-System aus der Perspektive der menschlichen Augen stereoskopisch aufgenommen. Die „Bildströme“ des Doppelkamerasystems werden durch ein Computersystem digitalisiert und in Echtzeit auf Texturen gezeichnet.

Der mittlere Teil der Abbildung („3D-Grafikszene“) zeigt die Projektion der beiden Texturen in eine virtuelle 3D-Grafikszene. In einer „normalen“ 3D-Anwendung würde diese Grafikszene aus der Sicht einer einzigen virtuellen Kamera aufgenommen. Das Ergebnis wäre, dass die Videostreams nebeneinander auf dem Computerbildschirm angeordnet würden. An dieser Stelle werden jedoch die Eigenschaften des Stereoskopie-Zusatztreibers der Grafikkarte ausgenutzt. Der Zusatztreiber erzwingt die Aufnahme der Szene aus der Perspektive einer zweiten virtuellen Kamera. Die beiden Kameras haben zueinander einen Abstand Δx. Δx simuliert den „Augenabstand“ und ist eine Konstante die im Konfigurationsmenü des Stereoskopie-Zusatztreibers festgelegt wird.

Die beiden virtuellen Kameras, welche die Grafikszene aufnehmen, haben eine exakte Entfernung und Ausrichtung zu den Projektionen der Videotexturen („virtuelle Leinwände“). Jede Kamera erfasst exakt das Abbild jeweils einer „virtuellen Leinwand“.

Es ist anzumerken, dass durch das beschriebene Verfahren keine zusätzliche Belastung des Hauptprozessors zu befürchten ist. Die zur stereoskopischen Darstellung der beschriebenen 3D-Grafikszene notwendigen Berechnungen werden praktisch vollständig vom Prozessor der Grafikkarte übernommen. Eine Prozessorlast entsteht lediglich durch die nicht zu vermeidende Konvertierung des Videodatenformats in das Direct-3D-Texturdatenformat.

Der Einsatz des beschriebenen Verfahrens in der Anwendung „Stereo View“ führte zu tadellosen Ergebnissen.

Manchem Leser wird die eingesetzte Methode ungewöhnlich oder gar umständlich erscheinen. Sie bietet jedoch gegenüber anderen Methoden gravierende Vorteile.

- Es wird eine relative Unabhängigkeit von der Grafikhardware erreicht[12].

[12] Alle Grafikkarten für die Stereoskopie-Zusatztreiber existieren können eingesetzt werden.

- Alle von der Grafikkarte unterstützten Stereoskopie-Videosignal-Formate können verwendet werden ohne Änderungen an der Anwendung vorzunehmen. Durch Umschalten des Signal-Formats im Konfigurationsmenü des Stereoskopie-Zusatztreibers können eine Vielzahl stereoskopischer Darstellungsverfahren umgesetzt werden. Beispielsweise das Anaglyphen-Verfahren zum Einsatz eines Standard-Videoprojektors mit Farbfilterbrillen oder das Polarisationsfilter-Verfahren zum Einsatz eines Doppelprojektor-Systems mit Polarisationsfilterbrillen.

- Die entstehende Software-Anwendung erreicht eine gute Kompatibilität zu unterschiedlichen Computersystemen, sowie eine hohe Stabilität und Robustheit, da mit der Direct-3D-Grafikschnittstelle eine ausgereifte Technologie zum Einsatz kommt.

In den folgenden Abschnitten wird gezeigt, wie das Darstellen stereoskopischer Videodaten über Direct-3D realisiert werden kann. Dazu wird neben der Initialisierung der Direct-3D-Schnittstelle das Einrichten einer entsprechenden 3D-Grafikszene und die Ausgabe von Videodaten auf ein stereoskopiefähiges Display dargelegt.

4.4.5.3 Direct-3D Initialisierung

Die Initialisierung der Direct-3D-Grafikschnittstelle findet in der Programmklasse *CDirect3D* der „Stereo View“-Anwendung durch Aufruf der Methode *CDirect3D::Init(...)* statt. Dier Erstellung einer Direct3D-Grafikschnittstelle ist in der Dokumentation von Direct X [13] und in der einschlägigen Literatur detailliert beschrieben. Dementsprechend wird dieser Prozess hier nur kurz und im Bezug auf die speziellen Anforderungen der „Stereo View“-Anwendung dargelegt.

Während der Initialisierung wird ein **Direct-3D Schnittstellen-Objekt** erzeugt, welches Informationen über die Grafik-Eigenschaften des vorliegenden Computersystems liefert. Abgefragt werden insbesondere die Fähigkeiten des installierten Grafikadapters. Dazu gehören unter anderem die möglichen Farbformate und Bildschirmauflösungen. Angelehnt an diese Fähigkeiten, wird eine **Direct-3D Geräte-Schnittstelle** erzeugt (Direct-3D Device). Diese Geräte-Schnittstelle repräsentiert im weiteren Programmablauf die Grafikkarte des Sys-

tems. Über die Schnittstelle werden spezielle Geräte-Einstellungen vorgenommen und Zeichenoperationen durchgeführt.

Die Initialisierung läuft demnach in drei Schritten ab:

1. Erstellung einer *IDirect3D9*-Schnittstelle zur Ermittlung der Fähigkeiten im System eingebauter Grafikhardware.

2. Festlegung der *D3DPRESENT_PARAMETERS*; diese legen den, auf die Eigenschaften der Grafikhardware angepassten Grafik-Modus fest.

3. Erstellung einer *IDirect3DDevice9*-Schnittstelle; diese Schnittstelle dient als Repräsentant der Grafikhardware unter denen in Schritt 2 festgelegten Eigenschaften.

Zu 1: Die Erstellung eines ***IDirect3D9***-Schnittstellenobjektes geschieht durch Aufruf der Direct X-Funktion *Direct3DCreate9(D3D_SDK_VERSION)*. Das Makro *D3D_SDK_VERSION* ist in der Direct-3D-Headerdatei *„d3d.h“* festgelegt und repräsentiert die Versionsnummer des verwendeten Direct X Software Development Kits. Die Funktion liefert als Rückgabewert einen Zeiger auf das Schnittstellenobjekt.

Zu 2: Über die in Schritt 1 erstellte Schnittstelle *IDirect3D9* können die Eigenschaften der Grafikhardware abgefragt werden. Angelehnt an die Fähigkeiten der Grafikkarte wird durch Ausfüllen einer Datenstruktur („*D3DPRESENT_PARAMETERS*“) ein Grafik-Modus festgelegt. Mittels der ausgefüllten Datenstruktur wird über eine Methode der *IDirect3D9*-Schnittstelle ein *IDirect3DDevice9*-Schnittstellenobjekt angelegt; es repräsentiert den fertig konfigurierten Grafikadapter und ermöglicht der Anwendung den Zugriff auf dieses Gerät.

In der Datenstruktur *D3DPRESENT_PARAMETERS* müssen zur Beschreibung des gewünschten Grafik-Modus eine Reihe von Parametern eingetragen werden. Im wesentlichen beziehen sich diese Parameter auf die Eigenschaften der zu verwendenden Bildpuffer. Festgelegt werden müssen die Ausmaße (Breite und Höhe), sowie das Farbformat der Bildpuffer. Abbildung 4.31 zeigt die in der „Stereo View“-Anwendung gewählten Einstellungen.

```
// create an empty structure

D3DPRESENT_PARAMETERS PParams;
ZeroMemory(&PParams, sizeof(PParams));

// fill structure with data

PParams.hDeviceWindow
    = hWnd;          // Handle to main-window

PParams.Windowed
    = bWindowed;     // Fullscreen or windowed mode

PParams.BackBufferWidth
    = SCR_WIDTH;     // Width of Screen/Window

PParams.BackBufferHeight
    = SCR_HEIGHT;    // Height of Screen/Window

PParams.BackBufferFormat
    = D3DFMT_A8R8G8B8; // Color-Format setting
```

Abbildung 4.31: Listing-„D3DPRESENT_PARAMETERS“

Zu 3: Die Erstellung der Geräte-Schnittstelle *IDirect3DDevice9* erfolgt durch Aufruf der Methode *IDirect3D9::CreateDevice(...)* des *IDirect3D9*-Schnittstellenobjekts. Abbildung 4.32 zeigt den Aufruf dieser Methode in der *CDirect3D*-Klasse der „Stereo View“-Anwendung.

Der *IDirect3D9::CreateDevice(...)*-Methode werden eine Reihe von Parametern übergeben. Der Parameter *D3DADAPTER_DEFAULT* weist Direct-3D an, den Default-Grafikadapter des Systems einzusetzen[13]. Der Parameter *D3DDEVTYPE_HAL* gibt an, welche Art Grafikadapter eingesetzt werden soll. Die Abkürzung „HAL“ steht für „Hardware Abstraction Layer“ und bezeichnet eine Grafikkarte mit hardwaregestützter 3D-Beschleunigung. Der Parameter *hwnd* ist lediglich ein Handle auf das Hauptfenster der Anwendung. Die nächsten beiden Flags stellen den Betriebsmodus der Direct-3D-Grafikschnittstelle für das zu erzeugende Geräte-Schnittstelle ein. Das Flag *D3DCREATE_HARDWARE_VERTEXPROCESSING* veranlasst die Berechnung aller geometrischen Transformationen durch den Prozessor der Grafik-

[13] Es könnten mehrere Grafikkarten im System eingebaut sein.

```
// create Direct3D-Device

  m_lpD3D->CreateDevice(
      D3DADAPTER_DEFAULT,
      D3DDEVTYPE_HAL,
      hWnd,
      D3DCREATE_HARDWARE_VERTEXPROCESSING
         | D3DCREATE_MULTITHREADED,
      &PParams,
      &lpD3DDevice9);
```

Abbildung 4.32: Listing-Erstellung *IDirect3DDevice9*-Schnittstelle

karte. Das Flag *D3DCREATE_MULTITHREADED* weist Direct-3D an, ein multithread-fähiges Device zu erstellen. Dadurch können mehrere Programm-Threads gleichzeitig mit der Geräte-Schnittstelle arbeiten. Im Falle der „Stereo View“-Anwendung konkurrieren drei Threads (die beiden Video-Textur-Filter und die Hauptanwendung) um die *IDirect3DDevice9*-Schnittstelle. Dadurch ist die Erstellung einer multithread-fähigen Geräte-Schnittstelle zwingend. Der Parameter *&PParams* ist die Adresse der in Schritt 2 angelegten und mit Daten gefüllten *D3DPRESENT_PARAMETERS*-Datenstruktur und beinhaltet den einzustellenden Grafik-Modus. Der letzte Parameter *&lpD3DDevice9* ist die Adresse eines noch „leeren“ Objektzeigers. Der Zeiger wird, durch den erfolgreichen Aufruf der *IDirect3D9::CreateDevice(...)*-Methode, mit der Adresse des neu angelegten *IDirect3DDevice9*-Schnittstellenobjektes gefüllt.

4.4.5.4 Direct-3D Transformationspipeline

Unter Transformation versteht man im Sinne der 3D-Computergrafik die Umformung und Anpassung geometrischer Daten (Vektoren). Darunter fällt das Verschieben, Drehen und Skalieren von 3D-Objekten, ebenso wie die Umformung von 3D Koordinaten in eine bildschirmkompatible 2D Form (ganzzahlige Bildschirmkoordinaten).

Die Transformation führt Direct-3D automatisch unter Zuhilfenahme einer **Transformationspipeline** durch und nutzt für die notwendigen Berechnungen, wenn möglich, die Grafikhardware. Die Transformationspipeline selbst besteht

aus drei „hintereinandergeschalteten“ **Transformationsmatrizen**: Die Weltmatrix, die Sichtmatrix (Kameramatrix) und die Projektionsmatrix.

Ist die Pipeline richtig konfiguriert, wird sie an ihrem Eingang mit dreidimensionalen Geometrie-Vektoren gespeist und liefert an ihrem Ausgang fertige, zweidimensionale Bildschirmkoordinaten.

Die **Welttransformation** wird durch die Weltmatrix durchgeführt. Sie ist dafür verantwortlich, dass ein zu zeichnendes 3D-Objekt verschoben, rotiert und skaliert werden kann. Die Weltmatrix entsteht in der Regel durch die Multiplikation einer Einheitsmatrix mit Translations-, Rotations- und Skalierungsmatrizen.

Die **Sichttransformation** sorgt dafür, dass eine 3D-Szene aus der Sicht einer virtuellen Kamera betrachtet werden kann. Für die virtuelle Kamera kann durch die Sichtmatrix eine Position und eine Blickrichtung im virtuellen 3D-Raum definiert werden.

Als letztes durchlaufen die 3D-Vektoren die **Projektionsmatrix**. Sie konvertiert die 3D-Vektoren zu 2D-Bildschirmkoordinaten und sorgt zugleich für eine korrekte Projektion, also für den „3D-Effekt“. Vereinfacht ausgedrückt bedeutet dies, Objekte nahe der virtuellen Kamera werden größer abgebildet, weiter entfernte kleiner. [2]

4.4.5.5 Einrichtung der 3D-Grafikszene

Die korrekte Einrichtung der **3D-Grafikszene** ist wesentlich für das Funktionieren der „Stereo View“-Anwendung. Eine 3D-Grafikszene besteht aus dreidimensionalen Modellen. Diese werden beschrieben durch Vektoren (geometrische Daten) und Texturen (Oberflächeneigenschaften). Zusätzlich muss eine virtuelle Kamera im Grafikraum platziert und eingestellt werden. Sie dient dazu, die 3D-Szene aus einer bestimmten Perspektive auf dem 2D-Display des Computers abzubilden. Das Ziel des noch zu beschreibenden Vorgehens ist die korrekte Abbildung beider Bildkanäle des Stereo-Kamerasystems auf den Augen des Betrachters unter Verwendung eines Stereoskopie-Zusatztreibers für die Direct-3D Grafikschnittstelle.

Das Einrichten der Grafikszene unterteilt sich in folgende Arbeitsschritte:

1. Festlegung von Position und Blickrichtung der virtuellen Kamera durch Definition einer Sichtmatrix (Kameramatrix).

2. Festlegung der Projektionsmatrix.

3. Berechnung von Position und Skalierungsfaktor der Videotexturen („Virtuelle Leinwände“) zur Aufnahme über die virtuellen Kamers.

Zu 1: Alle Vektoren einer dreidimensionalen Grafik werden zur Darstellung auf einem zweidimensionalen Display durch die Transformationspipeline der Direct-3D-Grafikschnittstelle geleitet. Die Pipeline besteht aus einer Reihe von Transformationsmatrizen, darunter die sogenannte **Sichtmatrix**, auch **„Kamera“-Matrix** genannt. Mithilfe der Kameramatrix lässt sich eine virtuelle Kamera - ein Beobachter - in eine 3D-Szene einfügen. Dazu muss die Position und Ausrichtung der Kamera definiert werden. Zur Ausrichtung gehört der Blickpunkt der Kamera (der Punkt, dem sie zugewandt ist) und die Ausrichtung der y-Achse ihres lokalen Koordinatensystems. Der Blickpunkt der Kamera wird durch den **„Look-At“-Vektor** ($\overrightarrow{la}$) festgelegt (Abbildung 4.33). Die Ausrichtung der y-Achse des lokalen Koordinatensystems wird durch den **„Up“-Vektor** ($\overrightarrow{up}$) beschrieben. Dieser besitzt in der Regel die Komponenten ($x = 0$; $y = 1$; $z = 0$). Andere Werte für den „Up“-Vektor würden ein „Rollen“ (Drehung um die lokale Z-Achse) der Kamera bewirken. Nachdem die Ausrichtung der Kamera durch Angabe des „Look-At“- und „Up“-Vektors definiert ist muss noch die Position der Kamera in der Grafikszene durch Angabe eines Positions-Vektors ($\overrightarrow{p}$) angegeben werden.

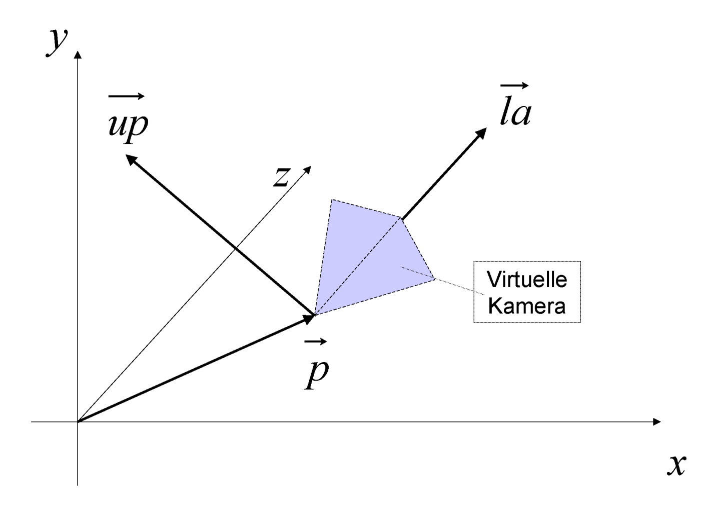


Abbildung 4.33: Virtuelle Kamera unter Direct-3D

Aus den drei Vektoren, die Position und Ausrichtung der virtuellen Kamera beschreiben, errechnet eine Direct-3D-Funktion eine Transformationsmatrix; die Kameramatrix.

Diese arbeitet nach folgendem Prinzip: Wird die virtuelle Kamera „verschoben" (Translation), so werden tatsächlich die Vektoren der Grafik verschoben und zwar genau in die entgegengesetzte Richtung. Wird die Kamera gedreht, werden die Vektoren der Grafik, mit der Kameraposition als Drehmittelpunkt, in die Gegenrichtung gedreht. Durch diese Vektoren-Transformationen entsteht der Eindruck, als befände sich ein Beobachter in der Szene. Dieser Beobachter ist dabei lediglich ein mathematisches Konstrukt.

Wie sieht es nun mit Position und Ausrichtung der virtuellen Kamera bezogen auf die „Stereo View"-Anwendung aus. Abbildung 4.34 zeigt die fertig konfigurierte Grafikszene in einer Draufsicht (Blick von „oben" auf die Z/X-Ebene). Die mit „Tex_L" bzw. „Tex_R" bezeichneten dicken Linien stellen die Videotexturen dar, auf denen die Videostreams der linken bzw. rechten Kamera des Doppelkamerasystems abgebildet werden. Die Einstellung der Direct-3D Kamera wurde mit „K" gekennzeichnet. Ihr „Look-At"-Vektor zeigt auf den Ursprung des Koordinatensystems (0; 0; 0). Ihre Position wird durch den Vektor (0; 0; $-a$) angegeben. Die Kamera wurde also, ausgehend vom Ursprung um einen kleinen Betrag a in negative Z-Richtung verschoben. Diese Szenen-Konfiguration würde ohne die Aktivierung der Stereoskopie-Zusatztreiber noch keinen Sinn ergeben. Auf dem Computerbildschirm würden nebeneinander Teile der linken und rechten Videotextur abgebildet werden. Durch die Aktivierung des Stereoskopie-Zusatztreibers geschieht nun folgendes: Die virtuelle Kamera wird zuerst von ihrer ursprünglichen Position um den Wert $-\frac{\Delta x}{2}$ „nach links" verschoben. An dieser Stelle wird eine Abbildung der Szene zur Darstellung auf dem linken Auge des Betrachters angefertigt. Wie auf der Zeichnung zu sehen, erfasst die Kamera K_L durch den eingestellten Blickwinkel exakt die linke Videotextur. Nachdem ein Abbild für das linke Auge angefertigt wurde, wird die Kamera von ihrer ursprünglichen Position um den Wert $+\frac{\Delta x}{2}$ „nach rechts" verschoben (K_R) und ein Abbild für das rechte Auge des Betrachters angefertigt. Die Konstante Δx beschreibt dabei einen, in der Konfiguration des Stereoskopie-Treibers einzustellenden, virtuellen Augenabstand.

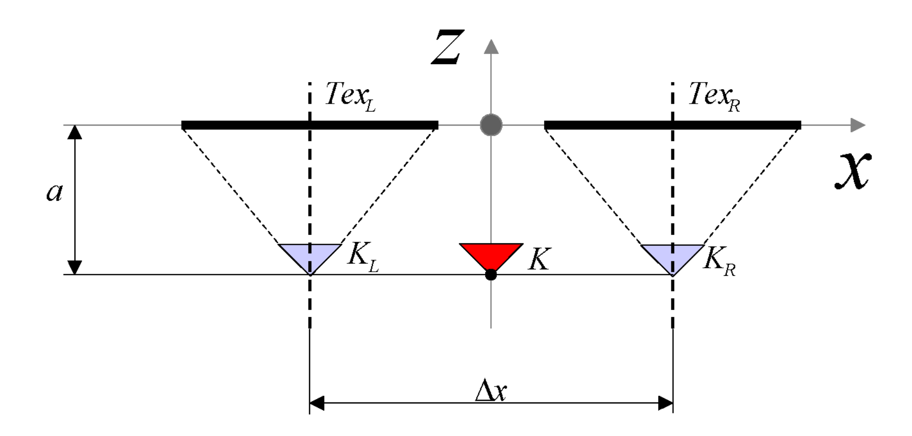


Abbildung 4.34: Kamerapositionierung

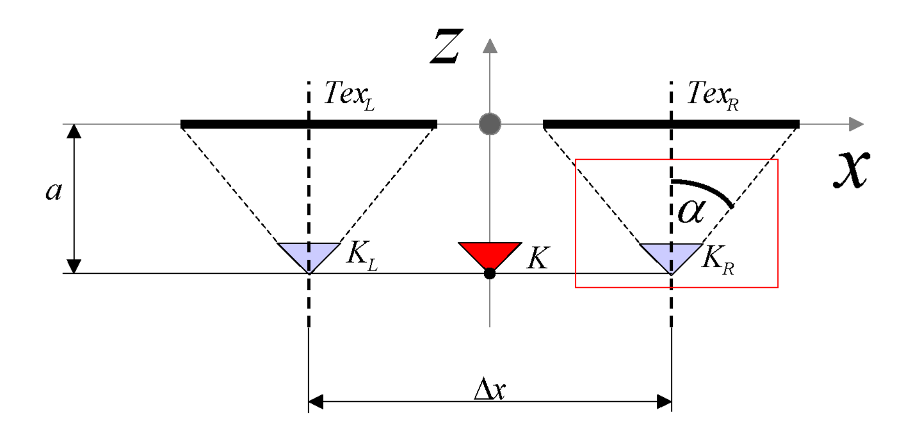


Abbildung 4.35: Definition eines Sichtfeldes (FOV)

Zu 2: Dem aufmerksamen Leser dürfte bereits aufgefallen sein, dass in Schritt 1 eine wichtige Eigenschaft der virtuellen Kamera noch nicht festgelegt wurde: Die Definition ihres **Sichtfeldes** (FOV: Field of vision). Die Angabe des Sichtfeldes geschieht in der Projektions-Matrix der Direct-3D-Transformationspipeline. Diese Matrix realisiert die Projektion dreidimensionaler Vektoren auf eine zweidimensionale Ebene (Bildschirm). Das Sichtfeld wird durch einen „Öffnungswinkel“ α definiert (siehe Abbildung 4.35).

Des weiteren wird durch die Projektionsmatrix eine nahe und eine ferne **Abschnittebene (Clippingplane)** definiert. Die Abschnittebenen sorgen dafür, dass Objekte die sich zu nahe vor (oder gar hinter) der Kamera befinden, sowie

Objekte, die sehr weit von der virtuellen Kamera entfernt liegen, nicht gezeichnet werden.

Als letzte Einstellung für die Projektionsmatrix muss das **Seitenverhältnis (Aspect Ratio)** des Sichtfeldes eingestellt werden. Es errechnet sich durch teilen der Bildbreite durch die Bildhöhe und beträgt für gewöhnliche Computerdisplays, als auch für das hier verwendete Stereodisplay, 4:3.

Die Berechnung der vollständigen Projektionsmatrix geschieht, ähnlich wie bei der Kameramatrix, durch Aufruf einer Direct-3D-Funktion unter Angabe von vier Float-Werten: Sichtfeldwinkel α (siehe Abbildung 4.35), Seitenverhältnis, nahe Abschnittebene und ferne Abschnittebene.

Für die „Stereo View"-Anwendung wurde ein Sichtfeldwinkel von 45° bzw. $\frac{\pi}{2}$ gewählt. Der gewählte Winkel vereinfacht die in Schritt 3 durchzuführenden Berechnungen.

Zu 3: Nachdem in den Schritten 1 und 2 die Eigenschaften und Position der virtuellen Kameras festgelegt wurden, müssen im letzten Schritt die Videotexturen skaliert und positioniert werden.

Die Positionen der beiden Videotexturen lassen sich sehr einfach bestimmen (siehe Abbildung 4.36). Die Texturen müssen zentrisch vor den beiden (durch den Stereoskopie-Zusatztreiber gegebenen) virtuellen Kameras K_L und K_R platziert werden. Die Kamera K_L ist an der Position $(-\frac{\Delta x}{2}, 0, -a)$ im Koordinatensystem platziert, die Kamera K_R an der Position $(+\frac{\Delta x}{2}, 0, -a)$. Um jede Videotextur zentrisch vor einer Kamera zu platzieren wird die Textur für den linken Bildkanal (Tex_L) vom Koordinatenursprung um den Wert $-\frac{\Delta x}{2}$ auf der X-Achse verschoben, während die Textur für den rechten Bildkanal (Tex_R) um $+\frac{\Delta x}{2}$ verschoben wird.

Die Texturen müssen aber noch auf die richtige Größe skaliert werden; sie sollen im Abstand a zu einer Kamera deren Sichtfeld gerade ausfüllen. Der Abstand a wurde, im Falle der „Stereo View"-Anwendung auf einen festen Wert von 0,28 Koordinatensystemeinheiten festgelegt. Mittels der bis dahin bekannten Einstellungen kann die erforderliche Breite einer Textur b berechnet werden:

$$b = \tan\alpha \cdot 2a = \tan 45° \cdot 2 \cdot 0,28 = 0,56$$

Die Videotexturen werden während des Zeichenvorgangs mit einem Skalierungsfaktor S auf die Breite b skaliert. Die Vertikalachsen der Texturen werden mit

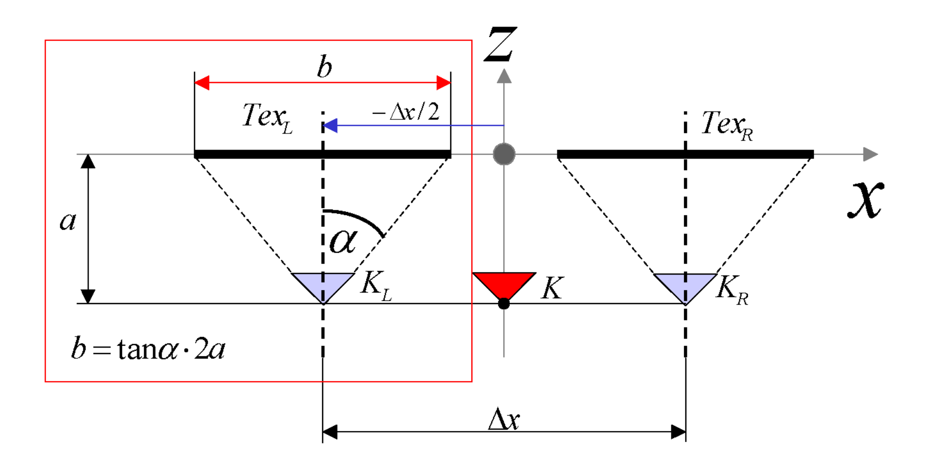


Abbildung 4.36: Positionierung/Skalierung der Videotexturen

dem gleichen Faktor ($S_y = S_x = S$) skaliert, so wird das Bildseitenverhältnis (4:3) beibehalten und die Textur erhält die richtige Höhe. Der Skalierungsfaktor S wird folgendermaßen berechnet:

$$S = \frac{Texturbreite_{gefordert}}{Texturbreite_{original}}$$

Die 3D-Grafikszene ist nun vollständig eingerichtet. Die stereoskopische Wiedergabe der Videostreams kann beginnen.

4.4.5.6 Zeichnen der Videotexturen (Rendering)

n den vorigen Abschnitten wurde durch die Initialisierung der Direct-3D-Grafikschnittstelle und das Einrichten einer 3D-Grafikszene die Eigenschaften eines virtuellen dreidimensionalen Raumes definiert. Auch wurde die Abbildung dieses Raumes auf dem Bildschirm des Computers durch die Definition einer virtuellen Kamera genau festgelegt.

Es können nun 3D-Grafikobjekte im Raum platziert und über die Grafikschnittstelle auf den Bildschirm gezeichnet („gerendert“) werden. Im Falle der „Stereo View“-Anwendung werden die zwei, vom Video-Texture-Rendering-Filter gelieferten Videotexturen über rechteckige Geometrien gelegt. Die daraus resultierenden Grafik-Objekte sind vergleichbar mit virtuellen Leinwänden. Auf diese werden die Bilder der (vom Doppelkamerasystems aufgenommenen) realen Szene projiziert.

Die „Stereo-View"-Anwendung aktualisiert in kurzen Abständen den Bildschirminhalt. Dazu wird das jeweils nächste Bild in einen leeren, sekundären Bildpuffer (Backbuffer) gezeichnet und während der Austastlücke des Videosignals auf dem Bildschirm präsentiert.

Eine Aktualisierung muss spätestens dann erfolgen, wenn einer der vom Doppelkamera-System eingehenden Videostreams ein neues Einzelbild liefert (Eingang eines Samples). Kameras, die Videosignale im PAL-Format liefern, haben eine Bildwiederholrate von 25 Vollbildern pro Sekunde. Bei gleichzeitiger Darstellung zweier, nicht synchronisierter, Videostreams muss also mit dem Eingang von bis zu 50 (2x25) Einzelbildern je Sekunde gerechnet werden.

Die „Stereo View"-Anwendung synchronisiert den Bildaufbau mit der vertikalen Bildwiederholfrequenz des Displays. Durch diese Technik wird ein eventuell auftretendes Bildflackern vollständig vermieden. Die vertikale Bildwiederholfrequenz beträgt, während des Betriebes des Stereoskopie-Zusatztreibers, aus Anwendungssicht etwa 50-60 Herz. Diese Frequenz übertrifft die maximale Anzahl pro Sekunde eintreffender Videobilder. Somit ist gewährleistet, dass jedes Einzelbild der beiden Videostreams gezeichnet wird.

Das Zeichnen der Videotexturen wird in der zentralen Programmschleife (*WinMain()*) der „Stereo View"-Anwendung durch Aufruf der Methode *Render()* der Klasse *CStereoCap* angestoßen. Abbildung 4.37 zeigt den prinzipiellen Aufbau der *Render()*-Funktion. Das Beispiel beschränkt sich auf das Zeichnen der Videotextur, die auf dem linken Auge des Betrachters abgebildet wird. Das Zeichnen der Textur für das rechte Auge erfolgt ähnlich. Das Grafikobjekt wird hier jedoch an der Position $(+\frac{\Delta x}{2}, 0, 0)$ (satt $(-\frac{\Delta x}{2}, 0, 0)$) im Koordinatensystems gezeichnet.

Die besagte *Render()*-Methode prüft nach ihrem Aufruf als erstes, ob eine der zu zeichnenden Texturen gerade, durch das Einspielen eines neuen Video-Einzelbildes, aktualisiert wird. Ist dies der Fall wartet die Methode auf den Abschluss dieses Vorgangs um nicht mit dem „Video-Texture-Rendering"-Filter zu kollidieren.

Anschließend wird eine Geometrie, in Form eines flachen Rahmens, erzeugt. Auf diesen Rahmen wird die Textur mathematisch „aufgespannt", damit Direct-3D sie darstellen kann[14].

[14]Direct-3D zeichnet im eingestellten Modus nur mit Texturen belegte Vektorgrafiken.

```
HRESULT CStereoCap::Render()
{
  // --- Zeichnen der linken Video-Textur

  // -- 1. Erstellung des Geometriepuffers

     SVertex aVertex[4] = { ... };

  // -- 2. Verschiebung der Videotextur auf der X-Achse

     const float excenter = DELTA_X/2;

  // -- 3. Warte auf Abschluss einer
  //       eventuellen Texture-Aktualisierung

     while(m_pTexfilterLeft->IsTextureLocked()) {}

  // -- 4. Videotextur fuer das linke Auge setzen

     g_Direct3D.GetDevice()->SetTexture(0,
                               m_pTexfilterLeft->GetTexture());

  // -- 5. Skalierungs- und Translationsmatrix
  //       erzeugen

     D3DXMatrixScaling(&mScale, 0.25f, 0.25f, 0.0f); // sx, sy, sz

     D3DXMatrixTranslation(&mTranslation,
                            -excent, 0.0f, 0.0f );

  // -- 6. Beide Matrizen zu einer
  //       Weltmatrix kombinieren

     mWorld = mTranslation*mScale;

  // -- 7. Weltmatrix der Transformationspipeline
  //       setzen

     g_Direct3D.GetDevice()->
               SetTransform(D3DTS_WORLD,
                            (D3DMATRIX*)(&mWorld));

  // -- 8. Als Dreiecksfolge zeichnen

     g_Direct3D.GetDevice()->
               DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,
                               2, aVertex, sizeof(SVertex));

  // --- Verfahre analogdazu mit der
  //     Video-Textur fuer das rechte Auge
  //     ...
}
```

Abbildung 4.37: Listing-Aufbau der Render-Funktion

Bevor eine Videotextur endgültig gezeichnet werden kann, muss die Weltmatrix der Transformationspipeline neu gesetzt werden. Sie sorgt für die richtige Positionierung und Skalierung des zu zeichnenden Grafikobjekts. Die Weltmatrix errechnet sich dementsprechend durch die Multiplikation einer Translationsmatrix mit einer Skalierungsmatrix. Die Translationsmatrix verschiebt die Videotextur an die Position $(-\frac{\Delta x}{2}, 0, 0)$. Vorher wird das Texturobjekt in x- und y-Achse um den Faktor $S = 0,25$ auf die passende Größe skaliert.

Zum Zeichnen des Texturobjektes wird auf das, bei der Initialisierung der Direct-3D-Schnittstelle erstellte, *D3DDevice9*-Schnittstellenobjekt zurückgegriffen. Es ermöglicht in der Anwendung den Zugriff auf die Funktionen des Grafikadapters. Die mit der Videotextur belegte Geometrie wird durch Aufruf der Methode *DrawPrimitiveUp(...)* in den Backbuffer gezeichnet.

Analog dazu wird mit dem Zeichnen der zweiten Videotextur (rechter Bildkanal) fortgefahren.

In der Austastlücke des Videosignals werden Front- und Backbuffer getauscht. Der Backbuffer wird zum Frontbuffer und die darin enthaltenen Daten werden auf dem Bildschirm angezeigt.

4.4.6 Leistungsmerkmale der Stereoview-Anwendung

Die „Stereo View“-Anwendung ist in der Lage, ausreichende Rechenleistung des Computers vorausgesetzt, die Bilddaten eines Doppelkamera-Systems in einer PAL-Auflösung von 720 x 576 Bildpunkten und einer Bildrate von 25 Vollbildern/Sekunde zu verarbeiten und auf einem stereoskopischen Display darzustellen. Das Bildverarbeitungssystem arbeitet somit „verlustfrei“.

Grundsätzlich können alle mit dem Betriebssystem Windows kompatiblen Videoeingabegeräte verwendet werden. Eine wichtige Voraussetzung zum Erreichen einer guten Bildqualität ist jedoch der Einsatz hochwertiger Kameras und Videocapturegeräte. Ebenfalls hat sich gezeigt, dass die Qualität des verwendeten Stereodisplays starken Einfluss auf die subjektiv wahrgenommene Bildqualität nimmt.

Kapitel 5

Schlussbetrachtungen

Zum Abschluss werden die behandelten Themen und erzielten Ergebnisse kurz zusammengefasst. Dem Leser werden, in Form eines Ausblicks, die Anwendungsmöglichkeiten des entwickelten Systems und zukünftige Forschungsideen nähergebracht.

5.1 Zusammenfassung

In dieser Arbeit wurde die Fähigkeit des räumlichen Sehens am Beispiel des Menschen erklärt und wesentliche Grundlagen zur Konstruktion stereoskopischer Aufnahme und Wiedergabesysteme vermittelt. Darüber hinaus wurde ein Konzept zur Konstruktion eines Stereokamerasystems für den mobilen Kleinroboter Sony AIBO ERS-7, sowie eines Fernsteuersystems für diesen Roboter erarbeitet und umgesetzt. Die Realisierung des Konzeptes geschah auf der Grundlage kostengünstiger Hardwarekomponenten. Beispielsweise werden die verwendeten Miniatur-Funkkameras üblicherweise in der Überwachungstechnik eingesetzt. Zusätzlich wurden neue mechanische und elektronische Bausteine konstruiert. Etwa eine kalibrierbare Halterung und Energieversorgung für das Doppelkamerasystem.

In späteren Kapiteln wird die Programmierung des Roboters AIBO ERS-7, unter Verwendung des AIBO Remote Frameworks, beschrieben und dem Leser die Realisierung eines Fernsteuersystems für diesen Roboter dargelegt.

Zum Abschluss der Arbeit wird die Implementierung einer Software zur Echtzeitverarbeitung und Darstellung stereoskopischer Videobilder geschildert. Dazu wird das Ansprechen von Videoeingabegeräten unter der Multimediaschnittstelle Direct X beschrieben und die Realisierung eigener Algorithmen zur Echtzeitverarbeitung von Videostreams erklärt. Ein neues Verfahren zur Darstellung der stereoskopischen Bilddaten eines Doppelkamerasystems unter Verwendung der Grafikschnittstelle Direct-3D wurde erarbeitet und realisiert.

5.2 Ausblick

Ziel der Arbeit war es nicht, ein System zur Lösung eines konkreten Problems zu entwerfen. Vielmehr sollte die Basis für eine ganze Reihe verschiedener Anwendungen geschaffen werden.

Bei den implementierten Funktionen handelt es sich tatsächlich um rudimentäre Fähigkeiten, die fast allen Lebewesen zukommen. In dieser Eigenschaft sind sie nicht dazu da, nur einzelne spezielle Probleme zu lösen. Vielmehr helfen sie in großem Umfang bei der Bewältigung der gesamten Erfahrungswelt. So sind die neuen Funktionen des AIBO nicht auf ein festes Problem gerichtet, sondern sie sind Instrumente, die dazu beitragen, unsere konkreten Sinneserfahrungen auf Maschinen zu übertragen, und diese so besser kontrollieren und steuern zu können. Damit scheinen sich tatsächlich eine Vielzahl praktischer Anwendungen realisieren zu lassen.

Sowohl die Software, als auch die Hardware ist modular und flexibel aufgebaut. Das Kamerasystem kann durch minimale Umbauten an der Kamerahalterung auf einer Vielzahl von Trägerrobotern eingesetzt werden; vor allem Klein- oder Flugroboter, die nur geringe Lasten tragen können, profitieren von der knapp 40 Gramm leichten, energiesparenden Einheit. Bei Austausch der Funkkameramodule gegen kabelgebundene Kameras wäre gar ein Einsatz des Systems unter Wasser oder unter der Erde, beispielsweise in Berg- oder Kanalbau, möglich.

Die Software des Fernsteuersystems wurde speziell für den AIBO ERS-7 Roboter ausgelegt und ist daher nicht ganz so flexibel auf andere Roboter übertragbar wie das Stereo-Kamerasystem. Dennoch bieten die erarbeiteten Konzepte interessante Grundlagen zur Steuerung von Telerobotern, auch wenn deren Anatomie von

der des AIBO abweicht. Insbesondere die präzise Steuerung einer Kamera durch Imitation der Kopfbewegungen ihres Bedieners findet viele Anwendungsfälle. Als Beispiel sei hier ein Arzt genannt, der über einen Teleroboter eine Operation durchführt; er könnte beide Hände zum Lenken der Operationsinstrumente verwenden, während die Kamera des Teleroboters automatisch seiner Blickrichtung folgt.

Das Stereo-Kamerasystem kann durch die Umsetzung weiterführender Forschungsideen in seinen Fähigkeiten noch wesentlich erweitert werden:

Beim Einsatz des Systems auf dem kleinen wackligen AIBO ERS-7 Roboter hat sich gezeigt, dass eine Stabilisierung des Kamerabildes die Qualität des Systems verbessern könnte. Eine solche Bildstabilisierung könnte beispielsweise in einem weiteren Schritt auf mechanischem Wege, durch eine Schwingungsentkoppelung der Kamerahalterung geschehen. Eine elektronische Bildstabilisierung über die Kamera-Software wäre ebenfalls denkbar. Diese würde aber, aufgrund der üblicherweise dazu verwendeten Technik, mit Verlusten in der Bildauflösung einhergehen.

Eine weitere interessante Möglichkeit des vorgestellten Stereo-Kamerasystems ist dessen Einsatz als eine Art optischer 3D-Scanner zur Erfassung eines Umgebungsmodells. Moderne Computer lassen heute eine mathematische Auswertung stereoskopischer Kameradaten in Echtzeit zu. Das Ergebnis einer solchen Bildanalyse wäre - berechnet aus zwei Teilbildern eines Stereobildes - ein einzelnes „dreidimensionales“ Bild, in dem jedem Pixel eine ungefähre Tiefenangabe zugeordnet werden könnte.[1] Durch Weiterverarbeitung der gewonnenen Tiefeninformation wäre es möglich mit hoher Geschwindigkeit ein ungefähres, dreidimensionales Modell der im Kamerasichtfeld liegenden Umgebung zu erstellen.

Kombiniert man ein solches topometrisches System mit anderen Sensoren, beispielsweise Ultraschall- oder Infrarotsensoren, verleiht man einem autonomen mobilen Roboter eine besondere Fähigkeit: Sich, ähnlich wie der Mensch, innerhalb sehr kurzer Zeit einen „dreidimensionalen“ Überblick über seine unmittelbare Umgebung zu verschaffen.

[1] Volker Hilsenstein präsentiert in seiner Dissertation [14] Evaluationen und Vergleiche von Algorithmen zur (Echtzeit-)/Analyse stereoskopischer Bilddaten.

Autonome Roboter, die unter höheren Geschwindigkeiten kollisionsfrei navigieren müssen, sind auf eine schnelle, dreidimensionale Erfassung der im Sichtbereich liegenden Umgebung angewiesen. Für sie brächte das zuvor beschriebene System erhebliche Vorteile. Als Beispiele seien hier autonome Fluggeräte (Helikopter, Flugzeuge) genannt. Unserem langsamen AIBO, der sich auf festem Untergrund bewegt, würde die Kollision mit einem Hindernis wahrscheinlich nicht schaden. Ein Fluggerät dagegen würde mit großer Wahrscheinlichkeit beschädigt oder so destabilisiert, dass ein Absturz nicht mehr vermieden werden kann.

5.3 Persönliches Schlusswort

Die Informatik bietet ein nahezu unerschöpfliches Spektrum an interessanten Bereichen, mit denen man sich beschäftigen kann. Jedoch ist die Robotik, meiner Ansicht nach, ein besonders interessantes Gebiet. Nirgendwo sonst bietet sich die Gelegenheit, die Fachgebiete Maschinenbau, Elektrotechnik und Informatik auf ähnlich spannende Weise zu kombinieren. Ich habe das Thema dieser Arbeit mit Begeisterung bearbeitet und die verbrachte Zeit sehr genossen. So hoffe ich, in der Zukunft die Gelegenheit zu haben, mich noch intensiver mit künstlichen Intelligenzen und der Konstruktion neuer Roboter auseinandersetzen zu können.

Ich hoffe, es ist mir gelungen mit meiner Arbeit einen kleinen Teil zur aktuellen Forschung beizutragen und interessierten Studenten und Wissenschaftlern die Möglichkeit zu geben, auf Grundlage meines Systems neue Forschungsideen zu evaluieren und umzusetzen.

Index

Literaturverzeichnis

[1] Mackenson/Von Hollander: *Das neue Wörter- & Fremdwörter-Buch*, XENOS Verlagsgesellschaft m.b.H. Hamburg, 1983.

[2] Scherfgen, David: *3D Spiele-Programmierung, (Modern Game Design with Direct X 9 and C++)*, Hanser, 2003.

[3] Stenzel, Roland: *Steuerungssrchitekturen für autonome mobile Roboter*, Dissertation, RWTH Aachen, 2002.

[4] Nasa: *Pathfinder Mission*, http://nssdc.gsfc.nasa.gov (03.01.2005).

[5] Sony: *AIBO ERS-7 Benutzerhandbuch*, Sony, 2003.

[6] Sony: *AIBO Entwicklerseiten*, http://openr.aibo.com/openr/ (03.01.2005).

[7] Prof. Dr. Michael Bach: *Räumlich durchs Auge.* In: c't 7/99, S.158, 1999.

[8] Studio 3D: *Stereo Video,* http://www.studio3d.com/images/ (03.01.2005).

[9] Kidsfactory, Japan: http://www.kidsfactory.co.jp/nuView/image/nuView.jpg (03.01.2005).

[10] Universität Erlangen, http://www2.chemie.uni-erlangen.de/ (04.01.2005)

[11] iO-Display Systems, Inc.: *I-Glasses*, http://www.i-glassesstore.com/ (03.01.2005).

[12] Sony: *AIBO Remote Framework Specification*, Sony, 2004.

[13] Microsoft: *Direct X Online Dokumentation*, Dokumentation zum Direct X-SDK, Microsoft, 2004.

[14] Hilsenstein, Volker: *Design and Implementation of a Passive Stereo-Infrared Imaging System for the Surface Reconstruction of Water Waves*, Dissertation, Universität Heidelberg, Fakultät für Physik und Astronomie, 2004, URN (NBN): „urn:nbn:de:bsz:16-opus-46018“.

[15] NVIDEA: *Display- und Stereotreiber*, http://www.nvidia.com (03.01.2005).

[16] Natural Point: *Track IR Head Tracking System*, http://www.naturalpoint.com/ (03.01.2005).

Eidesstattliche Erklärung

Hiermit erkläre ich, die vorliegende Diplomarbeit selbständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und habe wörtliche oder sinngemäße Zitate als solche gekennzeichnet.

Gießen, den Unterschrift: